HUMAN ERROR IN THE DESIGN OF A SAFETY-CRITICAL SYSTEM

by

Nick Shryane

A thesis submitted in fulfilment of the requirements for the degree of

Doctor of Philosophy

at

The University of Hull

September, 2003

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

ACKNOWLEDGMENTS

Thanks indeed are due to Dr. Martin Crawshaw and Prof. Bob Hockey for all their help and advice. Most importantly, I would like to thank Dr. Steve Westerman, my colleague on the EPSRC grant on which this thesis is based.

Thanks must also go to all the folk at G.E.C. Alsthom Signalling Ltd, Interlogic Control Engineering Ltd, Signalling Control UK Ltd, Sitec Ltd, and Westinghouse Signals Ltd. They put up with a lot of videoed intrusion of their work and answered many a stupid question.

I would also like to take the opportunity to thank God, just in case Pascal's wager also applies to viva voce examinations....

DEDICATION

Για την Πασμήνα μου...

1 Human error in the design of a safety-critical system

This thesis is an investigation into some of the causes and possible remedies to the problem of human error in a complex human-machine system. The system in question is engaged in the design of computer software for the control of railway signalling infrastructure. Error in its operation has the potential to be lethally destructive, a fact that provides not only the system's epithet but also the primary motivation and significance for its investigation.

The thesis consists broadly of two parts. The first part is concerned with exploration of the system, i.e. hypothesis generation, and is somewhat qualitative in nature. The second part is in general more quantitatively based, involved in the testing of the candidate hypotheses. However, it ends with a fairly lengthy departure from empiricism to consider some of the more fundamental issues raised but not answered by the work.

Because of the broad and exploratory nature of the initial investigation the themes on which it is based are quite general: 'human-machine systems' and 'human error'. The remainder of this chapter will expound these terms. More specific literatures will be discussed later, as and when required.

1.1 Human-Machine Systems

From bone tools to spacecraft, humans have a long history of using artefacts to help them achieve their goals. For a long period these artefacts were relatively simple and often used by their designers, promoting an easy mapping between intention and operation. Especially since the industrialisation of their production, however, artefacts have become much more complex and widely available, with a consequential decrease in their ready usability.

However, it took the necessities of world-war II to give birth to a discipline focused on why people found it difficult to use artefacts properly (for a brief history see Sanders and McCormick, 1993). At this early stage the discipline betrayed its narrow interests through one of its names; man-machine studies. However, not all humans are men,

and not all artefacts are machines. Further, consideration of a single person interacting with a single artefact is only a simple example of the general case, where complex systems require the interaction of many people with many types of artefact, both tangible and informational. The broadening of the name of the discipline to 'human-machine systems' still misleads somewhat, but it is in current use and will serve its purpose here.

At its basis a human-machine system can be thought of as any number of humans interacting with any number of artefacts in pursuit of a goal or goals. This rather broad definition will be structured further by way of Edwards' (1972) SHEL model. This describes systems in terms of their components, partitioned into the following four categories that make up the SHEL acronym:

- Software. Many aspects of a system are not physical in nature, but instead are 'virtual' or informational. The rules of arithmetic and the meaning of a clock-face display, for example, are instances of 'software'. The term also includes the definition that it has become synonymous with, i.e. computer programmes.
- Hardware. The physical aspects of the system, such as a computer mouse, or a doorway.
- Environment: This is that which surrounds and may influence the system, but which is not considered part of the system itself. Relevant aspects of the environment can be physical in nature, such as temperature and size, or more abstract, such as the economic environment. The environment can be considered to be the framework in which the system must operate.
- 'Liveware'. The humans that are part of the system, e.g. operators, managers; their characteristics and abilities.

The attribution of system elements to the various categories is a somewhat arbitrary exercise, particularly when it comes to defining the boundary between the system and the environment. For instance, consider a school. It is likely that teachers, pupils, caretakers, desks, toilets, books, numbers, roofs, emergency exit signs and electricity would all be considered necessary elements of a school system with the goal of

educating children in the UK today. However, what about the ministry of transport and the environmental health inspectorate? Currently the majority of staff and pupils must be able to get to a school, and that place must be reasonably safe from hazards. Both institutions have influence on the conduct of school business, but seem to be more distant from the core goal of educating pupils. For my purposes here the boundary between system and environment will be drawn based upon the smallest set of elements that are causally interactive. A teacher can have an effect on a pupil and vice versa, but a teacher cannot easily affect transport policy (although the converse is true).

Within this SHEL framework, problems arise when there is a lack of fit between the abilities of the liveware and the characteristics of the software and hardware. A 50 cm high doorway, for instance, would be unusable for a large proportion of the general public. Instructions for the use of a hand-dryer written in Polish would be fine in a Warsaw public toilet but be of less use, on average, in Seoul. The problem of physical correspondence between liveware and hardware, the domain of anthropometry, will not form part of this thesis, however. Rather, the fit between liveware and software will dominate – the realm of cognitive ergonomics.

1.1.1 Safety-critical systems

Safety-Critical Systems (SCSs) are those systems in which failure can lead to injury or loss of life; therefore it is in SCSs that the issue of error has perhaps the greatest importance. SCSs are often thought of as dealing with great physical forces or toxic materials, for example nuclear power stations (Health and Safety Executive, 1992), mass transportation systems (e.g. aviation: Weiner & Nagel, 1988) and medicine (e.g. Bogner, 1994). Indeed, the SCS studied in this thesis is a computer-based railway control system.

However, given the above definition of SCSs I would argue that this common view is too narrow. Much safety-critical activity is carried out in areas as diverse as social work (e.g. child protection: The Victoria Climbié Inquiry, 2003) and sport and leisure (e.g. in-line skating: Canadian Academy of Sport Medicine Sports Safety Committee, 1998), where poor systems also cost lives. Therefore, although this investigation is context-

bound by nature of the railway control system studied, the findings should be applicable to any system where error is of concern.

1.1.2 Reliability

The essential difference between safety critical- and non-safety critical systems is our attitude towards their reliability. With the cost of failure so high, SCSs must be as reliable as is practically possible. The concept of reliability is a central one to this thesis, so it seems sensible to discuss exactly what is meant by this.

Systems are implemented to carry out specific functions for a period of time. For example, the Humber Bridge was designed to transport road traffic across the Humber estuary until at least the year 2100. If it stops doing this before that date, i.e. it fails (e.g. physically, economically) through unintended acts or omissions, we could say that it has failed to meet its reliability target – it would be an unreliable bridge. In this case reliability is considered to be the probability of system failure within a given time span.

UK Def Stan 00-55 (Ministry of Defence, 1997) is a widely-used standard for safety-related computer software applications. For SCSs with the highest safety integrity level (level four, where system failure could be "catastrophic", resulting in multiple deaths), the probability of failure in any given year should be no greater than 1×10^{-4}, or one in ten thousand. In comparison, the Microsoft Windows™ Millennium Edition operating system running on the PC used to write this thesis fails (i.e. crashes, requiring a re-boot) about once every two days – around a million times less reliable than a level-four SCS. (This is actually an overestimate of its reliability because the computer is not in continuous operation.)

UK Def Stan 00-55 gives a reliability *target* for SCSs. A crucial factor not addressed by this standard, however, is how we should know when this target has been attained (or indeed, how we can attain it). In the example of the computer system (above) the estimate of reliability was based upon the observed behaviour of the system whilst in operation. This is clearly not an acceptable way to obtain reliability estimates for SCSs. Rather, reliability analyses are carried out that seek to predict the reliability of the whole system based upon knowledge of the observed reliability of individual components of

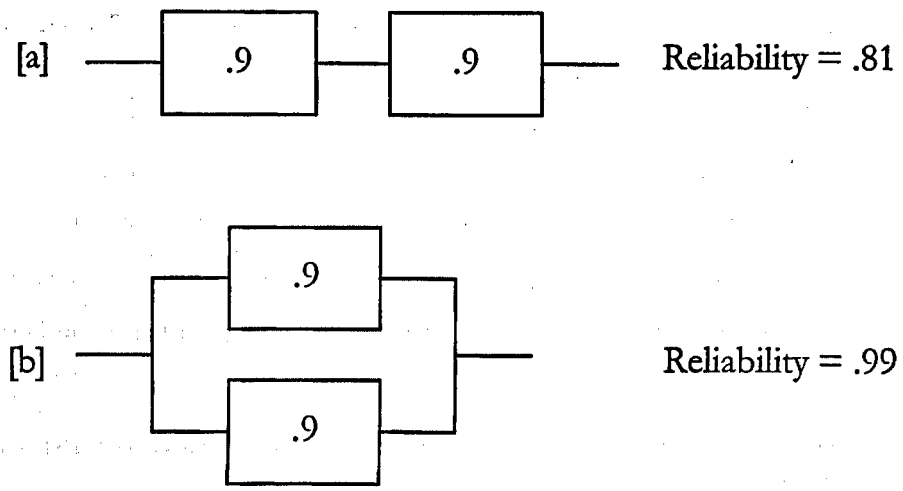


Figure 1: A system with two components arranged in series [a] or parallel [b]

that system. For example, consider the following example related by Wickens and Hollands (2000). It consists of a very simple system, with only two components, shown in Figure 1.

Each component has a probability of failure of .1, and therefore a reliability of .9 (reliability = 1 – probability of failure). If these two components are arranged in series [a], then if either component fails then the system as a whole will fail. The reliability of the system as a whole is .9 x .9 = .81. On the other hand, if the components are arranged in parallel [b] then the system will only fail if both of them fail together. The probability that the system will fail in this case is .1 x .1 = .01, giving a reliability of .99. When components of a system are arranged in parallel the system is said to have redundancy.

1.1.3 Human reliability

Human Reliability Analysis (HRA) is the extension of the above method into the domain of human system components (see, e.g. Miller and Swain, 1987). HRA seeks to decompose human task performance into a number of components to which error probabilities / reliabilities can be attached. These human error probabilities (HEPs) are

usually expressed in the form of a ratio of the occurrence of error divided by number of opportunities for error.

In doing so some methods rely upon databases of human error statistics (e.g. THERP; Miller and Swain, 1987; Swain and Guttman, 1983), while others rely upon expert judgement (Absolute Probability Judgement; see Kirwan, 1994). There are, however, a number of criticisms which have been made of the HRA approach to absolute error probability quantification (e.g., see Hollnagel, 1993; Miller and Swain, 1987; Rasmussen, 1986). It is certainly the case that HRA techniques are more dependable when comparing the relative error probabilities associated with different parts of a task than when used to give absolute error probabilities (HSC, 1991). Comparison of HRA techniques has revealed variance in absolute judgements of error probabilities "...between teams as well as within teams, typically with a factor of 10 or more" (Hollnagel, 1993, p. 132). There are a number of reasons why this might be the case. First, it can be argued that unique circumstances which prevail in each task environment mean that the statistics contained within error databases cannot be generalised with sufficient accuracy (see Taylor-Adams and Kirwan, 1995). Second, human error is often detected by the person carrying out the task him- or herself. Therefore, it is difficult to derive the probability of system failure simply from the probability of human error, as is discussed later in this chapter. The persistence of human error within complex systems relates to the nature of the task environment (i.e. opportunities for self-detection), and the self-detection of errors relates particularly to specific types of human error, as will also be discussed below. Third, a basic assumption of HRA methods is that human errors are independent of other human errors and of non-human system failures. This is demonstrably not the case, and furthermore the nature of this dependence is complex and difficult to specify (see Wickens, 1992, p. 432). However, some progress is made on this topic in § 9.3.2.

Even though absolute error probability judgements are problematic, I contend that the identification and classification of human error, along with the consideration of relative (within task) error probabilities remain worthwhile and attainable goals, and will be used in this thesis.

1.2 Human Error

The concept of human error seems intuitively important to reliability, and indeed will provide the focus of this thesis. What is meant by the term must therefore be made clear before proceeding, as it is easily confusable with other related but distinct concepts. The exposition below is informed greatly by the work of Norman (e.g. 1988) and Reason (e.g. 1990).

First, errors relate to a failure to achieve a goal. Without a goal there can be no error *per se*, and the same activities may be considered to be correct on one occasion and erroneous on another depending on the goal to be achieved. For example, deciding to clean my teeth might be considered a good thing shortly before going to bed, but an error while trying to drive safely along a motorway.

Second, because errors relate only to goals they also relate only to entities that can formulate and strive for goals. In human-machine systems the goals are set by humans, and so in this context the term 'human error' is somewhat of a tautology.

Third, errors relate to the cognitions and actions of people, but they are distinct from the consequences of these cognitions and actions. The activities of people may result in undesirable elements and states of systems, but these follow from the errors and are not the errors themselves. Specifically, **errors** may result in system **faults**, and system **faults** may lead to system **failures**.

To illustrate the above, consider the following example of a system. The system consists of a homeowner (the liveware), the homeowner's house with a newly installed alarm system (hardware), the instructions for using the alarm, written in English (software), and the environment, which in this case is considered to be the threat of burglary. Because of the threat of burglary the homeowner would like to protect his or her house from damage and theft. Therefore, he or she decides to activate the alarm every time the house is vacated for more than a few moments. On leaving the house the homeowner presses a sequence of buttons on the alarm control panel, derived from his or her understanding of the instructions, in the belief that the alarm will be activated. However, in **error** the homeowner merely goes through the test sequence

rather than activating the alarm. Therefore, when the house is unoccupied the alarm system is at **fault** in relation to the system goal (protect the house from theft and damage). However, although the homeowner continues to **erroneously** operate the alarm system in this way for many years, the **fault** never leads to system **failure** because the house is never burgled.

Not all systems can tolerate faults in this way without consistently leading to failure. If I fill the fuel tank of my car with diesel fuel rather than unleaded petrol, the fault of having a tank mostly full of diesel will quickly lead to the car's motive failure. (However, whether the decision or action of doing this would be considered an error or not will of course depend on whether my goal was to foul the engine.)

The extent to which faults lead quickly and consistently to failures can be though of as the 'coupling' between system elements (Perrow, 1984). In tightly coupled systems faults lead rapidly and/or inexorably to failure. A good example of a tightly coupled system is a house of cards, where virtually any fault, i.e. a fractional misalignment of a card, will lead to collapse. Notice that the fault could be introduced by error, e.g. clumsy placement of a card, or the decision to construct the house of cards in a draughty room, but this is not the only way for a fault to occur; an earth tremor would do the trick also, as would deliberate vandalism (a 'violation', as termed by Reason, 1990). On the other hand, in loosely coupled systems faults do not inevitably lead to system failure. For example, the internet was designed to be a loosely coupled system, able to tolerate many faults in its communications web and yet to continue functioning.

Faults do not always lead to failures, and similarly errors do not always lead to faults. The Greek astronomer Claudios Ptolomaios ('Ptolemy'), born c. A.D. 85, constructed a model of the solar system that was in use for approaching 1500 years. Given the goals and instruments of astronomy at the time (predicting planetary motions, eclipses, etc., all done without telescopes, or clocks as we now know them) it was a highly accurate model, which is for the most part why it persisted for so long. However, it was a geocentric model (i.e. with the earth at the centre of the solar system), a feature probably based primarily upon an erroneous and egocentric interpretation of the apparent motion of the sun. We now know that the reasoning that led to the model

was in error, but the model itself did not lead to any verifiable faults until more accurate data on planetary motions were produced by the Danish astronomer Tycho Brahe (b. 1546).

The detection of errors can come about before any fault or failure of a system, of course. For instance, motor control errors can be corrected very rapidly by comparing an 'efference copy' of the movement commands sent by the brain with a representation of the desired movement 'goal' (Kawato & Gomi, 1992; Rabbitt, 1978). In fact, this system is so fast that motor outputs can be modified or inhibited before they have had a chance to be put into action, i.e. errors are detected and corrected before they result in faults. More generally, Rizzo, Ferrante & Bagnara, (1994) described the concept of 'inner feedback' as information available to awareness/working memory that did not arise from external sources. These internal error-detection processes will not, however, be the focus of error-detection in this thesis. Rather, the processes by which faults are detected will be studied.

The detection of faults, by definition, requires feedback from the human-machine system external to the human operator. For instance, while writing this thesis I became aware of most of my typographical errors through the process of visual perception of misspelled words (or perception of spell-checking computer software cues), not through the internal monitoring of motor commands (although for a more skilled typist the reverse might have been true). Therefore, to detect faults depends upon our ability to perceive and distinguish between the actual state of the system and the desired state – to compare feedback with some representation of the goal.

The more 'direct' the comparison between existing and goal states the easier is likely to be the fault detection. If there is an accurate internal model of the goal state and ample, timely feedback about a relevant system state, then fault detection is fairly easy: the equation 7 + 2 = 10 should be a clear and available stimulus to readers who have been able to read this thesis thus far, but it does not seem to conform to the well-known rules of arithmetic. As such, it is a fault, and therefore suggests the presence of an error in the thinking or behaviour of its author, if his goal was to correctly sum seven and two. (I did intend it to demonstrate an error, which would make it a fault of arithmetic

but not of exposition. If I now claim that it is an equation written in base nine, rather than the expected and conventional base 10, then it is a correct example of arithmetic, but not of error...)

Clear and timely feedback is of no use for fault detection if the internal representation of the goal state is itself flawed, however. Consider the English word that means 'a person who owns or runs a restaurant'. Restauranteur? The word is actually 'restaurateur', with no 'n'. The former has become common, presumably because of its consistency with the spelling of the word 'restaurant'.

Without the perception of feedback on the mismatch between actual and desired system states, errors and faults will remain undetected (until and unless they lead to perceptible system failure). For example, Ptolemy's theory of the heavens was revealed to be faulty only when more precise feedback became available, in this case in the form of empirical data on the positions of heavenly bodies.

(Reason, 1990, referred to faults that lie dormant for some time without providing any feedback as to their existence as 'latent system errors', and faults that provide "immediate" feedback as 'active system errors'. I believe that Reason's scheme, while thematically consistent with the one described here, unnecessarily mixes notions of errors, faults and failures, and the coupling between them.)

Even when feedback from faults is available, however, it may not be interpreted as evidence of error, as the feedback itself may be dismissed as faulty. Improvements to Newton's theory of gravitation (1687) did not emerge from better observational data, but by theoretical inconsistencies between it and James Clerk Maxwell's (1873) theory of electromagnetism. This is despite the existence of observational data inconsistent with Newton's theory ('problems' with the orbit of mercury noted by Leverrier in 1855), but which were attributed to observational deficiencies. More recently, mounting evidence of faults in the thermal protection system of the NASA's space shuttle system did not lead the system's managers to react as though they had found evidence of system error until the catastrophic failure of the Columbia spacecraft in 2003 (Columbia Accident Investigation Board, 2003).

These issues of error feedback are basically those discussed by Quine (the 'Duhem-Quine thesis', 1961/1953) and Popper (e.g. 1972). Quine argued that it is impossible to view any piece of feedback as indicating unequivocally the presence of error in a system, because our understanding of the system can never be known to be complete. Popper argued that the existence of error feedback is the *only* way to conclusively demonstrate a faulty system. (Their topic was the practice of science, but I would argue that 'science' is a system containing humans and artefacts, like any other discussed here).

1.2.1 Error types

Many classification schemes have been proposed that seek to address issues of human error through the nature of faults. A well known example of a generic phenomenological error typology distinguishes between errors or faults of omission (a required act was not performed) and those of commission (incorrect performance of a required act or performance of an act which was not required). A number of authors have proposed more extensive phenomenological classifications. For example, Miller and Swain (1987) suggest that errors of commission can be further divided into selection errors, sequence errors, time errors, and qualitative errors. Hollnagel (1993) proposed a scheme in which four basic phenotypes are used as the basis of an attempt to model comprehensively all possible sequences of task performance: i) correct action - the correct sequence of steps; ii) jump forward - the action sequence jumps forward, missing steps from the correct sequence; iii) jump backward - the action sequence jumps backwards to already executed action steps; and iv) intrusion - an action is executed which is not part of the current plan.

These approaches are limited if only the faults themselves are known. For instance, consider the simple example of the formula above: $7 + 2 = 10$. What are the possible errors? Before making any suggestions, assumptions about the goal underlying performance must be made. Assuming that the goal was to write a correct equation in base 10 there are still myriad possibilities. There could have been an error of commission in the answer ($7 + 2 = 9$), but also in the numbers to be summed (e.g. $7 + 3 = 10$), as well as in the operators (e.g. $7 + 2 \neq 10$). There could instead have been

errors of omission (e.g. 7 + 2 = 10 -1). These examples hopefully serve to show the difficulties present when trying to infer errors from faults. However, knowledge of faults and failures is often the only source of information available when trying to investigate error.

1.2.2 Errors and expertise

In the earlier example, the goal of writing the word 'restaurateur' was initially failed and the word was misspelled. This could be seen as the commission of one error resulting in one fault – a spelling mistake. However, it could also be viewed as the writing of twelve letters correctly and the addition of an extra one erroneously. This is because virtually all goals can be thought of as being composed of numerous sub-goals (e.g. Newell and Simon, 1972). The extent to which we should consider it one or 12 goals will depend upon how these goals are represented internally by the typist.

A total novice at typing would probably have to treat typing each individual letter as a goal in itself, and perform each action in sequence to achieve the super-ordinate goal of typing the word. Performance would be slow, effortful and demanding of short-term memory resources (e.g. which letter to type next?) and conscious attention on the task. Errors would be quite likely and even more likely under stress, distraction or while performing other tasks concomitantly, especially those demanding similar, limited processing resources (e.g. Wickens, 1991). With practice the typing of individual letters would become chunked into typing groups of letters, until eventually even these chunks were integrated into an effortless ability to type the whole word with virtually no demands on short-term memory or conscious attention (Keele, 1968). Errors when performing such a skilled task are relatively infrequent, even while under stress and distraction (e.g. Tayyari and Smith, 1987).

This change in performance as skills are acquired has been extensively described: Fitts and Peterson (1964), Anderson (1982), Rasmussen (1983) and Reason (1990) have all described this process with models incorporating three levels of behavioural control. Performance at the novice level, called variously the 'cognitive' or 'knowledge-based' stage, is characterised by slow, consciously demanding and effortful performance. Partially-skilled behaviour has been described as the 'associative' or 'rule-based' level,

where aspects of the tasks can be performed without conscious attention, but these task chunks still need conscious triggering and linking. Finally, expert performance has been termed the 'autonomous' or 'skill-based' stage, where goal-directed behaviours can be performed without conscious attention. Performance at the novice level is slow and effortful but flexible and adaptable. In contrast, as performance becomes more learned and nearer to the expert level it is quick and effortless but rigid and stereotypical.

Regarding errors, Norman (1981) suggested a dichotomous classification based around whether the goal itself is selected in error or whether the action taken to achieve that goal is erroneous. On the one hand, if a sub-goal is selected that will not lead to fulfilment of the super-ordinate goal it is termed a 'mistake'. On the other hand, if the activity selected to achieve the goal or sub-goal is performed incorrectly (or not performed), this is termed a 'slip' (or 'lapse').

For example, I may have the goal of driving safely and efficiently from my home to my workplace. A sub-goal of this is to obey traffic signs and signals. Part of the route involves crossing a traffic-light controlled junction with two route options: straight ahead or a left turn, each controlled by its own traffic-light. Imagine that both lights are showing a red 'stop' signal, and that I am waiting to go straight on. I see the light in front of me change to amber then green, and I formulate the sub-goal (not necessarily with conscious awareness, if this is a well-practiced skill for me) to start on my way. I do so, and promptly crash into a car turning across my path, down the (for me) left-hand road. I had misperceived which traffic-light had turned green; in actuality it was the left-turn light that had changed and the straight ahead light was still on red. In this case I had made a mistake. I had formulated a goal (start driving) that, while performed correctly, was in conflict with some of my higher order goals, i.e. those of driving safely and obeying traffic signals. (It was consistent with another of my goals, however, that of driving from my home to my work.)

On the other hand, imagine me again waiting at the red traffic light. This time I perceive correctly my straight-ahead light change to green, and formulate the same goal as before, to move off and continue on my journey. However, by accident I select third

gear instead of first and stall the engine, jerking forward to a halt. The driver in the car behind, expecting to move forward, might even bump into the rear of my car. In this case the goal was correct but was not executed properly; I had made a slip.

It's also quite feasible that I could have made both errors together; misperceiving the traffic light and selecting the wrong gear. In this case the combination of both errors might have resulted in a more favourable outcome than if I had merely made either one; I wouldn't have crashed into the car in front, and the car behind, not expecting to move, would have been less likely to have bumped into my rear. (However, this would merely have been a fortuitous feature of the coupling between errors, faults and failures in this particular system; more errors and faults are usually worse for system reliability than are fewer.) In general, though, the occurrence of multiple errors and faults is to be expected in any reasonably complex system; "there's many a slip 'twixt the cup and the lip", as the saying goes.

Using a combination of Norman's (1981, 1988) and Rasmussen's (1983, 1986) models, Reason (1990) offered an integration and slight elaboration on the above, with his generic error modelling system. He proposed that slips and lapses were likely to dominate in the domain of expert, 'skill-based' behaviour. He also proposed that mistakes come in two varieties, depending upon the level of behavioural control. In fully consciously-controlled behaviour, the 'knowledge-based' domain of novice performance, mistakes of information processing occur, such as the overloading of short-term memory, confirmation bias (e.g. Wason and Johnson-Laird, 1972) and the like. In between these two endpoints, in the domain of 'rule-based' behaviour, mistakes of perception dominate, whereby otherwise efficacious chucks of behaviour are inappropriately triggered when the situation is misclassified.

In the first example from the driving scenario related earlier, a rule-based mistake was made when the driver misperceived the traffic light and initiated a skilled behavioural sequence that would have been appropriate in other circumstances. In the second case, the traffic light was perceived correctly and the correct goal, to start driving, was selected, but the execution of the skilled action was flawed.

1.3 Performance Shaping Factors

There are a number of factors which will influence the propensity of an individual for error. These are often termed Performance Shaping Factors (PSFs: Miller and Swain, 1987) within the context of HRA, and may be classified as either external (i.e. relating to characteristics of the task environment) or internal (relating to characteristics of the individual). Examples of external PSFs are: work layout, environmental conditions, work design, training, job aids, and supervision. In contrast, internal PSFs relate to the "...skills, abilities, and attitudes that the worker brings to the job" (Miller and Swain, 1987, p. 223).

In order to determine the influence of PSFs upon task performance they must be considered within the context of broader models of cognition. Although a comprehensive review is beyond the scope of this thesis, I will briefly mention a few of the most influential factors which are applied within this context.

1.3.1 Attentional limitations

Since the late 1950's, much research effort has been devoted to developing models of attentional limitations. Early work in this area focused upon attentional selectivity and was concerned with identifying 'bottlenecks' in the information processing system (e.g. Broadbent, 1958; Deutsch and Deutsch, 1963; Treisman, 1964). More recently, the ability to simultaneously process information has become the central focus and models which characterise attention as one (Kahneman, 1973; Moray, 1967) or more (Wickens, 1984) pool(s) of resources which can be flexibly allocated to task performance have enjoyed much success in predicting task performance.

The implications for human error are that if task demands exceed attentional processing capacity error is more probable. This may lead to "mistakes of bounded rationality" (Reason, 1987), which are reflected by simplified or incomplete planning of actions. Alternatively, "mistakes of reluctant rationality" may occur, as a result of individuals adopting task performance strategies which avoid novel thought (i.e. only considering a restricted problem space) in order to minimise 'cognitive strain' (Bruner, Goodnow, and Austin, 1956).

1.3.2 Mental models of the task environment

An influential theoretical approach which relates to the mental representation of the task environment is that of the mental model (Gentner and Stevens, 1983; Johnson-Laird, 1983). Mental model theories postulates that for any given task environment an individual will inevitably form a mental model which is "...a rich and elaborate structure, reflecting the user's understanding of what a system contains, how it works, and why it works that way" (Carroll and Olson, 1988, p. 51). This will be used as the basis for predicting the outcome of future interactions with the task environment. A mental model may be incomplete or inaccurate, and as a consequence "mistakes of imperfect rationality" may occur. These errors may be typified by "procedures that are too rule-bound, too rigid, and too conservative. Solutions to previous problems will continue to be applied and too little account will be taken of actual or potential change" (Reason, 1987, p. 18).

1.3.3 Individual differences

There are a number of dimensions of individual difference which will exert an influence upon task performance. These may relate to labile differences, such as mood or expertise, or to more stable differences, such as cognitive ability or personality (see van der Veer, 1989). Individual differences can be predicted to interact with the effects of each of the frameworks of cognition described above. For example, one of the main factors which will determine the availability of attentional resources is expertise. As individuals become more expert in task performance, so task performance becomes automated (see Schneider and Shiffrin, 1977) and attentional resource demands are reduced. This is analogous to the shift in performance level from Knowledge-based through to Skill-based as tasks become more practised and expertise is acquired. Similarly, expertise may reflect a more complete or more accurate mental model of the task environment which will serve to reduce errors.

1.4 Summary

To summarise, errors are failures in goal-directed activity, with a multitude of potential causes depending upon the knowledge, skills and abilities of the individual concerned. Errors often, but not always, introduce faults into the wider human-machine system,

and faults may lead to failure of the system in some way. Often a crucial goal of an SCS is seen as minimising error. Hopefully the preceding analysis has made it clear that, while this is desirable, the actual overriding goal should be that of minimising system failures. To understand how this may be achieved first it is necessary to understand what the system is composed of and how it is coupled together – how errors may lead to faults, and how faults can lead to failures. This task is tackled in the following chapter.

2 Task Analysis of the Solid State Interlocking system

This chapter presents a task analysis of the way in which software for a computer-based railway control system is produced. The control system is called Solid-State Interlocking (SSI), and its purpose is to allow safe and efficient movement of trains over a railway network. It will be described in more detail through the course of this chapter.

First, a process analysis (Piso, 1981) of the SSI system will be presented. This provides a description of the 'raw materials' for later analyses to work upon (e.g. what elements constitute the system?). Second, a Hierarchical Task Analysis (HTA; Annett & Duncan, 1967) will be shown, which decomposes the jobs that make up the activity of the SSI system.

2.1 Process analysis of the Solid State Interlocking system

Piso (1981) proposed that a process analysis is a useful and necessary exercise to conduct before the more formal Hierarchical Task Analysis (HTA). This is because HTA represents data primarily about the goals and actions of the human operators of a system, without explicitly describing what else the system consists of (e.g. software, hardware), its goals, how its components are arranged, the logic of its operation, and so on. The process analysis is intended to give the reader the necessary framework within which to view the subsequent task analyses.

2.1.1 Method

Piso (1981) suggests goals rather than methods should be used as a way to structure the analysis. The analysis should result in a description of what the system does and how it does it. As an extension of Piso's method the process analysis is structured here in terms of Edwards' (1972) SHEL model: Software, Hardware, Environment and Liveware. This extra categorisation provides more structure to the data and facilitates integration of information from the Process- and more complex Hierarchical Task-Analysis.

2.1.1.1 Sources of data for the process analysis

Piso (1981) suggests interviews with task experts as the sole source of information for the process analysis. This investigation used approximately five hours of interviews with four senior participants drawn from the two main industrial collaborators, but also task observations and task documentation. Documentary information was gathered from a number of sources: 'SSI 8500 - Design of SSI Signalling Schemes' gives a broad overview of railway signalling rationale and implementation; 'SSI 8003 - Data Preparation Guide' is the standard manual for the detailed design and coding of the SSI programming language. Both manuals are comprehensive in their coverage of material, extending to over 100 pages each.

The primary focus of this phase was not to examine the role of human designer. Neither was it intended to be a comprehensive description of the hardware and software components. For further details of these systems the reader is referred to Cribbens (1987) and Leach (1991).

2.1.2 Results

2.1.2.1 System goals

A railway network must satisfy two overriding goals:

1. It must be 'live', which means that it must allow trains to travel between points in the railway network. The more trains that can travel across the network at the same time, at higher speed and/or closer separation, the greater the volume of traffic the network can handle and so the greater the 'liveness' of the system.

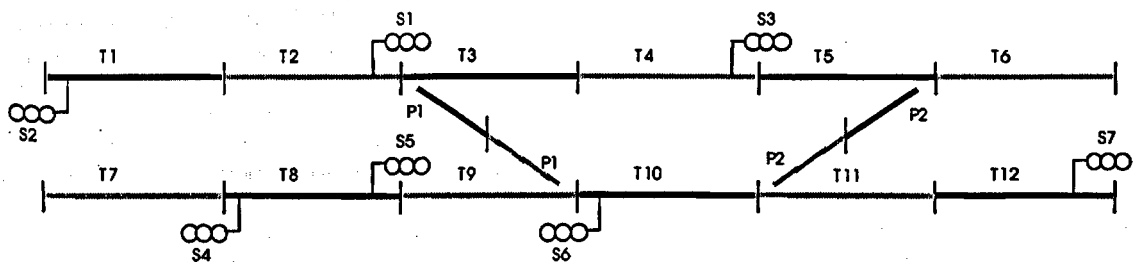
2. It must be safe. Trains must not be allowed to crash into one another or be derailed. Additionally, trackside workers and the general public must be protected or warned of approaching rail traffic when and where they are likely to come into conflict.

The way that the SSI system achieves the above conceptual goals is by satisfying a set of more concrete goals:

3. The SSI system maintains a real-time model of the state of the specific railway network under its control. It receives and stores information about train positions and the status of trackside equipment, e.g. points and signals, as well as information regarding the commands to alter the system state issued by human signal workers.

4. Based upon the information from the real-time model, the SSI controls the status of the trackside equipment under its command, allowing the system to only enter certain permissible states (designed to achieve 1 and 2, above).

Figure 2, below, shows a simplified railway signalling track diagram, with two main (horizontal) tracks connected by two short diagonal sections. Trains travel along the tracks from signal to signal. The SSI system must ensure that only safe train movements are allowed.



s = signal t = track section p = points

Figure 2: Schematic of an example rail network

As stated in goal 4., the SSI system must allow the railway network to enter only certain permissible states, and to stop it entering any others. It uses the information from the real-time model to feed into a control programme, which determines whether requests from human signallers or automatic timetabling software would put the system into an allowable or unallowable state. For example, a request to send two trains onto the same section of track at the same time should be refused by the SSI, as they would be in danger of crashing into one another.

Figure 3, below, shows an example of the "Geographic Data", as the site-specific computer programme is called, that would control some of the functionality of the example rail network shown in Figure 2. It shows the conditions that must be fulfilled before Route 2 (R2) can be set, allowing trains to travel from Signal S1 to Signal S7. This route involves a train starting on track section T2, moving onto track T3 and down across the points onto track T9, then on via T10, T11 and T12 to signal S7.

```
*QR2      R2 a                          / Route label and availability
          P1 crf , P2 cnf               / Points test
          U10-AB f , U3-BC f            / Opposing route(s) test
     then R2 s                          / Route setting
          U3-CB 1 , U9-CA 1 , U10BA 1 ,  / Sub-route locking
          11-BA 1  U12-BA 1
          P1 cr , P2 cn                 / Points controlling
          S2 clear bpull                / Signal clearing
```

Figure 3: Example SSI Geographic Data

This entails checking that the route is available (R2 a), e.g. not barred because of maintenance; that the points are in the correct position, or it is safe to move them to the correct position, to allow the train to cross from T3 to T9 (P1 crf , P2 cnf); and that other, conflicting routes are not already set, which is done by checking two opposing 'sub-routes' to ensure that they are free (U10-AB f, U3-BC f). If these checks are passed, then the route is set (R2 s); the individual sub-routes in Route 2 are 'locked' (U3-CB 1, U9-CA 1, etc.); the points are moved to the correct position (P1 cr, P2 cn); and the route entrance signal is checked to see if it is 'clear' to be changed to green (S2 clear bpull). In the final line shown in Figure 3, signal S2 is commanded to change to green, an incorrect command as route R2 runs from signal S1. This error would be potentially disastrous, allowing two trains travelling in opposite directions onto the same section of track.

It is the programming, verification and validation process for this 'geographic data' that is the focus of the present study. The SHEL model will be used to structure the various elements that combine to achieve the above goals of the system.

2.1.2.2 Environment

The environments in which the system operates have an over-arching effect on the choices for how it should function. The commercial-economic environment will tend to value liveness over safety concerns. The social-legal environment will tend to value safety over liveness. This competition between goals may seem tenuously related to the question of how the SSI system achieves goals 1 and 2; however, this tension is likely to have direct impact on the general approaches to control used by the system, and so the methods that can be adopted to ensure the railway's functioning.

An example of this was seen in the wake of the 1988 Clapham rail disaster. It was claimed that Automatic Train Protection (ATP) systems, not in use at the time (although available), would have prevented the accident and 34 fatalities. ATP systems seek to keep a minimum separation distance between trains, as opposed to the current "absolute block" system which divides the rail network into sections which are only supposed to contain a single train at any one time. However, analyses which sought to enumerate the maximum amount of money a population (of rail users in this case) would be willing to pay to prevent each fatality revealed a maximum estimate of around £2 million. ATP would have cost at least £15 million (per fatality) to install (Hope, 1992; Jones-Lee & Loomes, 1995.) and so it's introduction, though promised by regulators, was dropped.

The economic and political climate has also forced radical organisational change onto the UK rail industry over the past few years. Now defunct, British Rail was a single, nationally-owned organisation responsible for all major aspects of the rail service. It therefore had responsibility and control over both safety and liveness, and could strategically manage the tension between the two. The situation today, however, is much more organisationally complex. There are many and various for-profit companies that between them provide rail transportation in the UK. The way that this has been organised has meant that responsibility and control for safety goals and

liveness goals do not lie wholly within the same organisation. Indeed, virtually any aspect of the operation of the rail network in the UK today relies upon the interaction of many separate organisations, e.g. train operating companies, signalling companies, engineering companies, all with sub-contractors and suppliers. (This situation, however, is now changing again, with the demise of Railtrack, the company that was previously in charge of rail infrastructure.)

The physical environment of the UK's rail network is compact and dense. Although reduced in overall size in the last 50 years, in places (e.g. London, Crewe) the network is still one of the densest in the world. This has meant that the SSI system must be complex and flexible enough to deal with the intricacy of the rail network it must control.

Regarding the physical environment in which SSI design work is conducted, it is carried out predominately in open-plan office-based environments. This means that designers have easy access to one another (and their expertise), but they also have only partial control over unwanted noise and distractions.

2.1.2.3 Hardware

2.1.2.4 SSI installation

The safety-critical processing of the 'geographic data' is carried out by three computer micro-processors. Each contains a validated generic control program and location-specific "geographic data" stored in Electronic Programmable Read-Only Memory (EPROM). The control programme, standard to all SSIs, interprets the geographic data. The geographic data is unique to each installation, however; each railway layout is as different as, say, the road layout in each town or city. This geographic data specifies information about the layout of the rail network and also the logic of the train movements that are, and are not, allowed. Each SSI processor maintains a real-time "map" of the state of the railway in Random Access Memory (RAM), including such information as the state of each signal and set of points.

Each SSI system contains three identical control processors which operate on a majority voting system. This is to guard against any one processor having control of

the system should one of the processors fail or have to be taken off-line for maintenance. This defends against mechanical failures and corrupted memory, but will not guard against incorrect geographic data, as the same version is loaded into each of the three processors.

2.1.2.4.1 The Design Workstation

The critical task of writing and validating the geographical data for each installation is carried out using the 'design workstation' (DWS). The DWS comprises a workstation computer (with a display screen, keyboard and mouse), and an SSI simulator with two colour display terminals, each with a trackball controller. 'Geographic data', the SSI computer code, is written using the UNIX-like DWS computer, which offers text-editing applications, compilers and so forth. The code can then be uploaded to the SSI simulator for testing. (The SSI simulator is essentially a cut-down SSI installation, with only one control processor instead of three, as 100% availability is not critical during design.) The SSI simulator is fitted with RAM instead of the usual EPROM to allow quick loading of newly written SSI code.

One of the simulator display screens presents a representation of the simulated state of trackside components (e.g. points, signals). The other screen shows a signaller's panel, which presents the controls and displays regarding the requesting and setting of routes and related functions.

2.1.2.5 Software

The most important non-physical elements of the SSI system are the Standard Signalling Principles' (SSPs) and the SSI geographic data 'language'.

The SSPs embody the philosophy of how the safety goals of system operation are to be achieved. The philosophy of 'absolute block' signalling is in use in the UK today, and has been for over 50 years. The basic tenet is that only one train is allowed into any one section of track at a particular time. The entrance into each track section is guarded by a signal which operates in a similar way to a traffic light. A green light means the track section is clear and trains can enter, a red light means the section is

occupied and the train must stop. That way if a train were to break down the last signal passed by the train would be on red, barring the route to following trains.

The geographic data language is a proprietary programming language specific to SSI. It was developed to have similar concepts and operational logic to the electro-mechanical relay signalling systems it replaced, so allowing signalling engineers with expert knowledge and skills in the previous system to be able to convert efficiently to SSI. Example SSI geographic data 'code' is shown in Figure 3; the similarity to generic, text-based programming languages is evident.

2.1.2.6 Liveware

The signalling engineers, at least those working for the firms participating in this project, were all male and from the UK ethnic majority. They tended to be from science and engineering educational backgrounds and all seemed familiar and competent with standard computer technology. They were heterogeneous with respect to their ages and levels of SSI knowledge and experience, however.

2.2 Hierarchical Task Analysis of the Solid State Interlocking system

There is a wide range of task analytic techniques which may be used to describe and evaluate human-machine and human-human systems, each with associated strengths and weaknesses (cf. Diaper, 1989; Kirwan and Ainsworth, 1992). Within the present context, a number of varied, and potentially conflicting, demands were placed upon the task analytic process.

The task of SSI data design incorporates a diverse range of component elements. It was necessary that the task analytic method be capable of describing this 'macro' structure of the design process, including the rules governing the overt behaviour of the designer. However, it was also important that specific components of the SSI data design process could be described in fine detail. In order to meet these potentially conflicting requirements of breadth and depth Hierarchical Task Analysis (HTA) was used.

HTA was originally specified in 1967 by Annett and Duncan at the University of Hull and has since been developed and refined by other workers (e.g. Piso, 1981; Patrick, Spurgeon and Shepherd, 1985). There were a number of reasons to believe that HTA would be the most appropriate technique to provide a description of the SSI design process. HTA is a versatile and powerful tool which has been applied in areas as wide ranging as the allocation of automation within human-machine systems (Fewins, Mitchell and Williams, 1992), operational safety assessment (Rycraft, Brown and Leckey, 1992) and the development and planning of training provision (Patrick et al., 1985). It provides a logical, hierarchical breakdown process which enables tasks to be specified in great detail. However, unlike other potential analysis methods such as Task-Action Grammar (Payne & Green, 1986), it is not so formalised that it becomes unworkable for very large tasks (such as SSI design).

In doing so, HTA focuses on the goals that the system is trying to achieve in terms of the activities that the human must perform to attain those goals. Thus, the activities of the human designer are linked directly to the systems requirements.

2.2.1 The HTA process

HTA seeks to break down an overall system goal into a hierarchy of sub-goals, together with the attendant plans and operations required to achieve them within task constraints. It seems pertinent at this point to discuss exactly what these terms mean within HTA, before going on to describe the procedure adopted for the analysis.

The basic concepts used in HTA are:

- **Goals.** HTA describes work activity in terms of the goals that are to be achieved. Goals in this context are related to attaining desired states of the system under control or supervision.

- **Tasks.** The task is the means of achieving a goal. There may be a number of different tasks that can achieve the same goal. Various constraints, such as time, availability of resources (e.g. trained personnel), and habit will influence which specific task is selected on any given occasion.

- **Operations.** Operations refer to units of behaviour which must be carried out in order to carry out a task. They specify the lower level "action-information-feedback" loops that make up controlled activity. It is at this level that aspects of performance difficulty can be assessed. For example, by examining whether the feedback from an operation is unambiguous.

- **Plans.** Plans refer to the circumstances under which various operations should be carried out. It is as important knowing when to carry out an operation as knowing what exactly to do.

The HTA analyst takes an arbitrarily-selected overarching goal and decomposes it into several sub-goals. A plan for scheduling the sub-goals is also derived, so that together the sub-goals and plan can be considered equivalent to the superordinate goal. Each sub-goal then becomes a candidate for decomposition into further sub-goals or operations. This process of decomposition could potentially continue *ad finitum*.

Therefore, at some point the level of detail of description of the sub-goals and operations involved must be assessed to see if they are adequate. This decision is aided by the use of a stopping rule. For instance, a common rule is the "PxC" rule. In this case, the probability (P) that the operation would be carried out unsatisfactorily is multiplied by the cost (C) to the system if this occurs. If the product is low the analysis of that particular branch of the hierarchy is stopped, if high, the analysis is continued to pinpoint the exact area of difficulty. Other stopping rules can be used and in this way the analysis can be tailored to the overall goal of the research.

2.2.2 Method

2.2.2.1 Sources of data

There are a number of ways in which data can be collected for HTA. Where available, documentation associated with the task, e.g. manuals or training information can provide detailed task specifications. Observation of the task can be particularly useful for detecting task organisation and scheduling. Interviews with task experts, however, provide the most flexible form of data collection, allowing the analysis to proceed as required at either the macro or micro level. A combination of methods can give a more accurate result than would be possible with just a single method. The current study employed interviewing as the major form of data collection, supplemented by observation and documentation.

2.2.2.2 Participants

Nine signalling engineers, from GEC Alsthom Signalling Ltd., Westinghouse Signals Ltd and British Rail Engineering, took part in the analysis. All were male and between 30 and 55 years of age. The bulk of interviewing (seven out of sixteen interviews) took place with the three most experienced engineers, all with over ten years of signalling experience. Each interview took place in a quiet room away from the interviewee's normal work area and lasted between approximately one and three hours.

2.2.2.3 Procedure

Initially, a statement specifying the goal that the SSI designer (task-expert) must achieve was elicited. The task-expert was then asked to restate the goal in terms of a number of sub-goals or operations, together with a plan for carrying them out. These

sub-goals, if performed according to the stated plan, should together be equivalent to the super-ordinate goal. Sub-goals were then assessed to decide if any of them required re-description at a more detailed level. Any sub-goals that required re-description were then treated as the super-ordinate, and the task expert was asked again to re-describe them in terms of a number of sub-operations together with the plan for carrying them out. This process was iterated until all the relevant areas of the hierarchy had been adequately described.

When considering the point at which the re-description should stop two principles were applied. The first was "PxC", as related previously. In instances where there was a difference of opinion between task experts when using the "PxC" rule an additional guiding principle was used. This concerned the degree to which task performance was concerned with skilled motor or cognitive operations. When tasks were concerned only with fundamental skills, e.g. moving a mouse pointer or reading a manual, they were not considered priorities for re-description.

As the task analysis progressed a semi-structured interview schedule was used to ensure that all pertinent areas in terms of the task analysis were covered. Areas of questioning included: the information used by operators in decision making; the manner in which feedback about actions and general system state are conveyed to the operator, and; potential problems or errors. After the first interview subsequent participants were shown the current analysis at the start of their interview. They were taken through it and asked to make comments, highlighting areas of disagreement or which they thought required clarification. This confirmatory approach allowed the analysis to continue to a greater depth than would otherwise have been possible with the available resources if the whole analysis was repeated at each interview.

2.2.3 Results

The hierarchical goal structure of the task of "Producing a commissioned SSI scheme" (Goal 0) is rather large, and so is reproduced in full in Appendix A. To aid the following discussion the first two levels are shown in Figure 4.

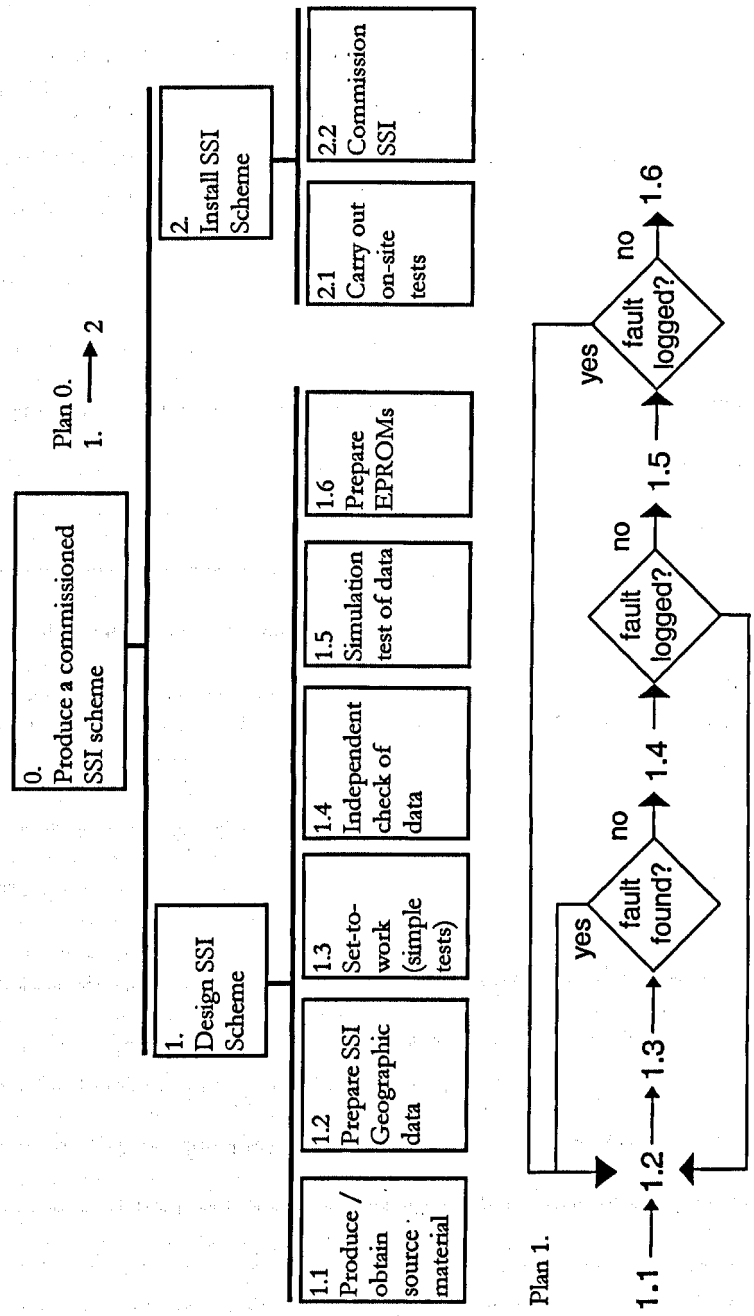


Figure 4: Partial Hierarchical Task Analysis of the Solid State Interlocking Geographical Data design process.

This HTA relates to the production of a single SSI. As can be seen, the overall goal can be subdivided into two second level subordinate goals, the first of which comprises an office based design process using the design workstation (Goal 1), and the second relates to the installation, checking and final commissioning of the SSI on-site (Goal 2).

Given the previously stated aims of this research project, it is the former of these two phases which is of primary interest, and for this reason the level of analytical detail is greater in this area of the task. However, it should be noted that many areas of the on-site checking process repeat earlier office-based checking stages (Goals 1.4 and 1.5) and many on-site task components could be expanded by the inclusion of some of these, previously described, elements.

The plan associated with Goal 0, indicates simply that office-based preparation is followed by on-site installation. However, it should be noted that, as described in Plan 1.2, errors in the on-site phase may require that the design process return to an earlier stage.

The first stage in the preparation of SSI geographic data requires that the appropriate source materials be obtained, or produced if they are not already in existence. These materials include: a copy of the operators requirements; the Standard Signalling Principles (SSPs); the Data Preparation Guide (a.k.a. SSI 8003); the scheme plan; a route list; and may also include control tables. Control tables are lists of the formal properties required from the interlocking and are produced from an analysis of the scheme plan. They are essential for the later checking and testing stages of the design process, but the point at which they are produced may vary according to the complexity of the scheme plan and the experience of the signalling engineer. Experienced signalling engineers may prefer to work from primary sources of information (i.e. the scheme plan) rather than introduce a secondary source (the control tables) at this point in the process which could merely propagate errors in the control tables. In this case the control tables may only be prepared immediately prior to the formal checking of the data.

Similarly, the preparation of the TFM database (a computer file which records the allocation of trackside equipment to parts of the SSI system) is included as a subordinate of Goal 1.1, as the contents are used in later stages of the design process as 'source information'. This decision is somewhat arbitrary, and it would also be reasonable to view the preparation of this file as a subordinate of Goal 1.2. However, the constraints of the task are such that the TFM database must be prepared before the geographic data files are written.

The preparation of the geographic data (Goal 1.2) requires that a number of identity files (Goal 1.2.1), data files (Goal 1.2.3), and panel files (1.2.4) are written. In addition the simulation screens must be prepared and co-ordinated (matched to the components of the particular SSI).

Identity files define unique names which are used to represent the component elements of the railway (signals, points, track circuits, etc.), or which reference bits in memory ('flags') which are used to retain information as to the state of the railway (e.g. timing information).

Data files contain statements defining the logic which will be applied to the operation of the railway. The IPT (input) and OPT (output) files control the input and output signals from the SSI processors. The FOP (Flag OPerations) file is primarily concerned with dealing with the aforementioned 'flags' in memory that retain system state information for future processing.

As the system must maintain a real-time map of the state of the railway network the information contained in the above three data files is processed regularly (about every about half a second). The information contained in the MAP (defining the elements of the railway, e.g. signals), PFM (which contain the logic for deciding if the Points are Free to Move to different positions), and PRR (Panel Route Requests, as set manually on the signaller's screen or automatically by timetabling software) is only processed as required. Panel data files contain information relating to the signaller's display screen.

All of these files may be produced using the DWS, which runs under a UNIX-like operating system. However, many of the engineers dislike the file editing facilities which are afforded and opt to write files using a PC system, and port these files to the UNIX platform. File templates may be used to speed the more repetitive elements of geographic data preparation. The data in these files may be regularly compiled in order to detect syntax errors.

As indicated by Plan 1.2 the sequence in which these sub-goals are performed broadly requires that the identity files be prepared first, followed by the preparation of the simulation screens. The preparation of the data files and panel files can then proceed in any order. The engineer may exercise some discretion over this sequence, choosing to write each file in stages, or to return to earlier stages in order to deal with complexities as they arise. Compilation of the files may occur at any point in this process, but must be the last operation performed in the completion of Goal 1.2.

The preparation of geographic data is followed by a process known as 'set to run' (Goal 1.3) in which the accuracy of specific component elements of the information contained in the data files is tested by the same design engineer who wrote them, on the SSI simulator. This takes the form of an iterative process in which, as errors are detected, the operator will return to the appropriate subordinate of Goal 1.2 in order to make the required corrections (see Plan 1). The tasks associated with this goal are in some respects similar to those associated with Goal 1.5 (Simulation test data). The main difference is in terms of comprehensiveness; the set-to-run is less stringent, and requires only that points and routes can be set, and that signals can show a green light when required. However, it may be that, at the discretion of the engineer, further testing is completed at this stage.

A new Central Interlocking Status Record (CISR) must be created before the data is passed on for checking. This provides version control for the data, and will follow the data as it goes through the checking and testing processes, providing a record of corrections.

Goal 1.4 takes the form of an independent check of the geographic data code. This is performed by a different, and usually more experienced signalling engineer. A paper copy of each of the files generated at Goal 1.2 is systematically checked against the source information (control tables and signalling plans) for faults. If faults are detected at this stage they are logged and, once the check is complete, details are returned to the engineer(s) who completed the initial preparation work, for correction.

The independent simulation test of the data (Goal 1.5) is again conducted by a further one or two highly experienced signalling engineers. This stage of the design process consists of checking the working of the SSI using simulation screens to represent signaller's and trackside information. A complex sequence of testing is undertaken in which the SSI is examined in relation to both the correct operation of the component elements of the railway, and also the functional requirements of the system.

The first stage of this process involves a correspondence test (Goal 1.5.1), in which the components on the signaller's and trackside displays are tested to see if they are correctly bound together (e.g. moving a particular set of points on the signaller's screen should result in the trackside display showing the movement of the same set of points). It was reported that faults detected at this point are most frequently associated with the misallocation of screen co-ordinates.

Correspondence testing is followed by 'principles testing' (Goal 1.5.2), in which the logic associated with the functioning of the railway is examined. The use of control tables is fundamental to this process. Faults are logged by the engineer, or engineers, performing the simulation test, and details are returned to the engineer who completed the initial preparation work (Goal 1.2) for correction.

Once these independent checking stages have been successfully completed the EPROMs are prepared (Goal 1.6). These chips contain a permanent record of the data for installation on-site.

The on-site testing (Goal 2) has much in common with the simulation-testing stages previously described, as the on-site SSI is run in 'simulation' mode (Goal 2.1). As with the earlier simulation test, a correspondence test is the first subordinate goal (Goal 2.1.1), and this requires that information on the signaller's display corresponds to the state of trackside equipment. However, if the interlocking is a replacement for a running system it may not be possible to incorporate the real trackside components in the testing process, for safety reasons associated with the concurrent running of the railway. In this case trackside simulations of signals, points, etc. are used in order to verify the correct transmission of information.

The on-site principles test (Goal 2.1.2) essentially covers the elements which could not be achieved by the office-based simulation test (Goal 1.5), and includes elements such as checking for lamp failures (Goal 2.1.2.1), timing problems (2.1.2.2), and complex cross boundary operations (2.1.2.3) where the SSI system must communicate with neighbouring railway areas.

The final commissioning of the SSI (Goal 2.2) involves the completion of the necessary paperwork, certifying the correct operation of the scheme, and handing over to the operator.

2.2.4 Discussion

As stated earlier above, the aim of this phase of the project was to provide a descriptive framework upon which to base subsequent field- and laboratory work. The primary areas of interest will be the programming of the geographical data files (subgoals of 1.2), 'set to run' (Goal 1.3), desktop checking (Goal 1.4), and simulation testing (Goal 1.5). These components form the kernel of the knowledge and skills required in designing and writing reliable SSI code. From now on, these tasks will together be referred-to as the SSI Data Preparation Process, or DPP.

Even regarding this circumscribed task domain there is still more detail available than the limits of this thesis would allow full treatment of. What follows therefore is discussion of a number of important points which arose during the course of HTA, some of which are used to guide further investigations in later chapters.

2.2.4.1 Staff allocation

The allocation of individual members of staff to the various elements of the DPP is largely based upon experience. The least experienced design engineers are required to write the data files. Some of these files are more difficult to program than others, for instance more experienced engineers may prepare the OPT and PRR files. More experienced engineers will also perform the desktop checking process, and the most experienced engineers will perform the simulation testing. Ideally this system of staff allocation provides an opportunity for training to take place, with less experienced engineers learning from the errors which are detected by their more experienced colleagues during the checking and testing phases.

There are a number of implications of this staff allocation policy with respect to the efficiency of the DPP and the forthcoming error analysis in the next Chapter. First, although the DPP is time consuming, it must be recognised that this is in part due to training concerns. An opportunity is being provided for less experienced engineers to develop their skills, and, as a consequence, it is probable that more errors will be made in the stages of the DPP where comparatively less experienced engineers are employed. Obviously, if this staff allocation policy were not adopted, the changed demands placed upon the experienced engineers and the suitability of alternative training methods would need to be considered.

Second, if the independence of the checking and testing is to be maintained, this policy requires that any assistance given to the less experienced engineers in order to correct errors is not provided by the person performing the checking or testing. Allied to this, care must be taken that 'in house' conceptual errors are not propagated (cf. Cutler, 1991).

Third, in these circumstances it will be difficult to determine whether a particular type of error is resistant to detection at a particular stage of the DPP because of the type of task demands, task environment, or because of the level of experience of the engineer.

2.2.4.2 Automatic Data Preparation

A current area of interest, which is strongly related to the issues above, concerns attempts which are being made to automate the DPP. Work has been undertaken at the University of Warwick (Collyer and Wong, 1993), and is in progress in each of the major organisations involved in SSI, to automate some or all of the tasks involved in the DPP. Each of these systems of automation appears to differ from the others in significant respects. However, it is recognised that 100% automation of the DPP is not currently realistic, and is perhaps undesirable (cf. Dennien and Needle, 1991), and that 80-90% might be a more achievable figure. The remaining 10-20% would comprise the most complex elements of the task and, as a consequence, time savings associated with automation may be of the order of 50%.

Information relating to the development of 'automatic data preparation' in each of the collaborating organisations was subject to confidentiality agreements, and specific details cannot, therefore, be included in this thesis. However, a number of potential problems arise from the change to partial automation of the DPP.

The automated DPP will require fewer signalling engineers, but they will need to be highly skilled and capable of dealing with the most complex task components. However, the opportunities to develop these skills on the simple parts of the DPP will be reduced. As a consequence, it will become more difficult to train engineers to the required level.

Allied to these changes in the role of the design engineer, the automation of the DPP may give rise to changes in the engineer's mental model of the system (see § 1.3.2) due to changes in the opportunities for information acquisition (Satchell, 1993). In certain circumstances automation has also been found to give rise to 'complacency' (Parasuraman, Molloy, and Singh, 1993), such that, for example, overconfidence in the automated processes may lead to comparatively simple errors being missed at the early stages of checking.

Given these developments, it may be that some elements of the existing training process, which incorporates the 'on-task' incremental development of expertise, will no

longer be feasible and that new methods of training must be investigated. One such possibility would be Computer-Assisted Instruction (CAI). CAI provides the opportunity to present material to trainees in a manner which can be tailored to the needs of the individual. 'Closed loop' methods can be used in which new learning can be coached and tested through interaction with the computer application. Training engineers in the use of the computer programming language elements of the DPP could quite conceivably be accomplished in this way. In a variety of domains CAI has been found to provide efficiency gains in training, with reduced training costs resulting from a reduction in training time and reduced demand for expert assistance (Eberts and Brock, 1988). The systematic decomposition of the SSI data design process which was achieved using HTA would lend itself to representation in a computer-based hypertext, and with some further elaboration of the task components, this might be developed to form the basis of a CAI training package.

2.2.4.3 Task Similarity and Diversity

The general structure of the HTA hierarchical diagram can be used to compare different aspects of the task being studied. For instance, the pattern of the overall diagram and the specific goals within this pattern indicate the similarity of the writing and checking tasks (Goals 1.2 and 1.4) compared to the set to run and testing tasks (Goals 1.3 and 1.5). Both pairs of goals share certain common elements within the pair that are diverse across pairs. For instance, the task environment (and perhaps mental models engendered by the task environment) is similar within pairs. When performing writing and checking the engineers are dealing with an abstract, symbolic representation of the SSI system, consisting of flags, sub-routes, and so on. When performing a "set-to-run" or functional testing the representation is very much more concrete, or analogue, where the engineer is dealing with actual railway concepts such as tracks and signals. This diversity in task environment will form a major part of this thesis in later chapters, and will be discussed in more depth there (starting in § 6).

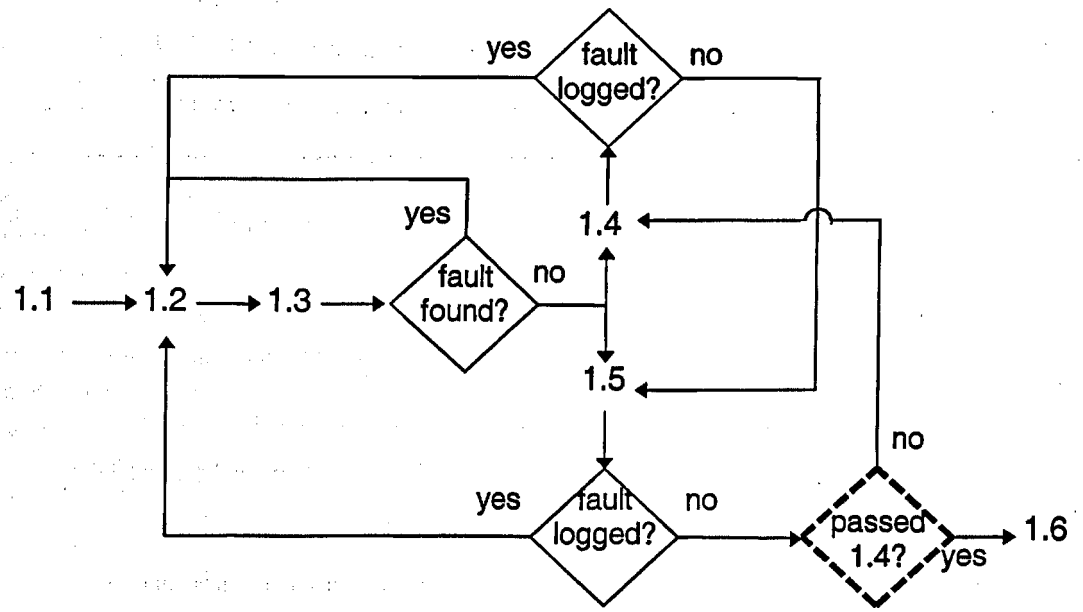


Figure 4: Revised plan 1, for when checking (1.4) and testing (1.5) are carried out in parallel.

2.2.4.4 Environmental influences

As discussed earlier, aspects of the environment in which the system operates can influence the way in which tasks are performed. One such influence was noted in relation to the way that urgent work is carried out.

Time pressure occasionally forces some checking and testing to be carried out in parallel, leading to a revised plan 1 (see Figure 4). This means that the version control for the data must be very tight, or unchecked data could be signed-off as safe by the tester. Normally, each new version of the data is given a unique version number by the data writer (Central Interlocking Status Record; CISR). This number records how many cycles of checking and testing the data has gone through, but not whether the latest version was generated because faults were found in a check or a test. If it was a test, then has that version of the geographical data been checked as being error-free before? The danger point is shown by the dashed lozenge in Figure 4. If this decision is made incorrectly then unchecked data could be released into service. This problem is exacerbated by the contracting-out of the checking or testing of these 'rush' jobs to other signalling firms, with an attendant increase in the difficulty of version control.

2.2.5 Summary

The SSI DPP was revealed as a complex task involving multiple staff members. It can be characterised as collaborative software production, but with the extra requirement of strict independence between processes to avoid the propagation of design errors through the production process.

The SSI DPP consists of three main tasks that operate two major processes that rest across two work 'domains'. The tasks are writing, checking and testing. Regarding work processes, there is the distinction between generation (writing) and validation (checking/testing). Regarding the domains of operation of these processes, writing/checking involves the use of computer programme code, and testing involves the use of a simulator to directly interact with SSI system behaviour.

Although the SSI DPP is a collaborative process the channels of communication between writer, checker and tester are strictly limited, occurring through formalised error/fault logs in an effort to allow only communication about the *presence* of a perceived problem.

The process- and task-analyses provided essential information regarding 'structural' properties of the SSI DPP. In the following chapter this will be supplemented by information on the dependability of the tasks and processes thus described.

3 Error Analysis of the Solid State Interlocking system

The HTA provided a useful framework for the breakdown of potential problem areas in the SSI DPP. As discussed previously, structural elements of the hierarchical diagram can be used to show similarities and differences between tasks. However, these similarities do not necessarily equate to similarities in actual task performance. For example, although identified as similar by the HTA, performance in writing and checking may not be identical even given identical requirements, i.e. code that is difficult to write may be easy to check and vice versa. What HTA does not reveal is how all of the variables that may affect task performance will actually combine to produce error.

This chapter, then, will focus on exactly which errors and faults *are* observed in actual SSI DPP tasks. Not only the individual tasks in isolation, but also the combination of tasks that together make up the overall system needed to be assessed. The aim of the error analysis chapter will be to describe the types of faults which are generated, the stages within the design process at which they are most prevalent, and the likely causes for these faults, i.e. error.

Several complementary techniques were chosen for the error analysis. These were chosen partly on the basis of availability, but also to give a broad range in terms of the type of data they would provide. They were for the most part based on techniques drawn from Human Reliability Assessment (HRA).

While the application of Human Reliability Assessment (HRA) techniques may prove useful in the identification of human error and the assessment of relative error probabilities, as discussed earlier it is contended that the generation of absolute Human Error Probabilities (HEPs) is not a worthwhile goal in this context. There are a number of further reasons for this. The nature of the SSI DPP task is such (complex information processing) that it is probable that many errors will be mistakes (rule- or knowledge-based errors). As discussed in § 1.2, it is a much more difficult proposition to quantify these errors on the basis of just fault information than it is to quantify skill-based errors. Furthermore, with respect to the SSI DPP task, there are many features

of the task which are unique (e.g. the SSI 'geographic data' language). As a consequence the utility of generalised HEPs, derived from human reliability databases will be limited. This is not to say, however, that HRA is not useful within this context. What is at issue is the ability of HRA to produce absolute estimates of the probability of human error. What is not at issue is the importance of identifying the characteristics of human error and its causes, in order that human reliability can be improved.

To investigate errors committed during the writing phase of the DPP a number of techniques were used. First, an SSI writer's work sample test was devised, to provide some control over extraneous variables that may affect the production of errors in actual task performance. Task observation with subsequent video walkthrough was also used to provide some measure of errors made but subsequently corrected by the writer himself.

The checking and testing phases of the DPP task were investigated using logs of actual faults found during the production of finished SSI schemes.

Further to this, data relating to errors generated at each stage of the SSI design task were gathered using structured interviews of task experts and from existing documentation. Each of these areas of investigation will now be discussed.

3.1 Work sample of the Data writing task

In order to make a more detailed assessment of the errors which are made during the DPP writing task a work-based, controlled experiment was conducted in which SSI design engineers completed a work sample. This technique presented the opportunity to investigate error in a manner which was, like the purely observational work, ecologically valid, but which also controlled for some of the extraneous and confounding factors present in a real work situation.

The specific task chosen was based upon the job content domain (e.g. Guion, 1988) of the writer's task as previously described in the HTA. The test content domain was selected in collaboration with the most senior SSI engineer available (who had participated in the development of the SSI system during the 1980s). A previously

completed set of SSI code files (for an SSI currently in service) had sections removed and the participants' task was to reinstate the missing code.

The primary aim for the work sample test was to provide some measure of the type and number of faults committed by writers while controlling for the complexity of the work. The deleted sections of code were chosen based upon Rasmussen's (1983; 1986) model of skill-, rule-, and knowledge-based control of behaviour. First, the greatest part of the task was selected from 'run-of-the-mill' code with which participants would be familiar, and which would be amenable to rule-based performance; Second, code was selected that controlled a function unique to the particular SSI scheme chosen, which would require participants to devise a novel solution, and therefore would require knowledge-based performance.

(It was decided that skill-based aspects of the writing task could not be easily isolated and mapped to sections of the SSI code in the way described above, because skill-based behaviour would underlie much if not all of the work that would need to be performed in order to carry out the rule- and knowledge-based sections; for instance, reading manuals, working with the keyboard, operating the SSI workstation.)

The second goal of this work was to garner some measure of the effect of individual differences among SSI engineers on task performance. A number of studies have found that variation in cognitive ability (particularly spatial ability) is associated with variation in performance across many computer-based tasks, e.g. word processing (Gomez, Egan & Bowers, 1986), information retrieval (Vicente, Hayes & Williges, 1987), and programming (Foreman, 1988). There is less clear evidence in favour of the importance of personality in computer-based tasks, although Pocius (1991) concluded that introverts tended to be better programmers on average than did extroverts.

3.1.1 Method

3.1.1.1 Participants

Fifteen SSI engineers were recruited from two organisations (three sites in total). All were men aged between 20 and 46 years (M = 32 years). (Although this is a small

sample, fifteen engineers represents perhaps as many as a third of the qualified participants working in the UK at the time of data collection.)

3.1.1.2 Materials

To ensure veracity, production of the work sample task was carried out by a highly experienced SSI engineer. The data files from an in-service SSI (Gerrard's Cross) were specially prepared with sections missing. The incomplete code consisted of 1006 lines in total, with the absent sections (which needed to be replaced) amounting to 124 lines. The code corresponding to rule-based performance consisted of 120 lines; the remaining four lines corresponded to knowledge-based performance.

Participants were provided with all the usual supporting documentation for carrying out the SSI DPP (see § 2.1.2.5 for details). Task performance was video recorded with two VHS video cameras; one camera was focused upon the screen of the Design Workstation and recorded interaction with the computer. The second camera was focused upon the surrounding desk area and recorded more general activities (e.g. breaks from work activity).

The same engineer who designed the work sample task produced the necessary materials and procedures to ensure the accurate scoring of the participants' solutions to the task. This consisted of a copy of the complete and correct SSI source code that formed the basis of the task and a procedure to have the DWS produce a "difference file" that showed all the areas where the participants' solutions differed from the original, correct version.

Prior to task performance participants completed a number of measures of individual difference. These included:

- PREVUE (Bartram, 1994): This test provides data on four 'higher order' personality factors: Independence; Conscientiousness; Extraversion; and Stability. In addition, eight 'lower order' factors (two for each of the main factors) are measured.

- A battery of cognitive ability measures drawn from the General Aptitude Test Battery (U.S. Department of Labor, 1982). The specific tests were: Name Comparison (clerical perception); Three-Dimensional Space; Vocabulary; Tool Matching; and Arithmetic Reasoning.
- The computer literacy subtest of the Computer Aptitude, Literacy, and Interest Profile (Poplin, Drew, and Gable, 1984). This is a 30 item test of computer-related semantic knowledge.
- A questionnaire asking about previous education and SSI signalling experience.

In addition, participants completed the UWIST Mood Adjective Checklist (UMACL: Matthews, Jones, & Chaimberlain, 1990) before and after task performance. This self-report measure produces a three factor solution to mood, comprising: energetic arousal; tense arousal, and; hedonic tone.

3.1.1.3 Procedure

The work sample test was conducted at the participant's place of work, in a room containing a DWS that had been set aside for the sole use of the study. Participants first completed the set of questionnaires described above. They then had a five minute break before commencing with the work sample task. The work sample materials contained all of the instructions necessary to complete the work sample test. Participants were asked to complete the task as quickly and accurately as they could. They were asked to repair the code to the point that they would normally complete an initial "set-to-run" (an informal simulation test of the code to see if it works, see § 2.2.3 for more details).

3.1.2 Results and discussion

The timing and sequence data were prepared for analysis by viewing the video recordings and using purpose designed software which allowed the timing and sequence of specific events to be logged by pressing marked computer keys. In particular, occasions when reference to documentation was made; time spent typing; time spent writing notes and time on break, were recorded.

At the end of each test the participant's SSI data file was compared to the correct version and a list of differences was produced. These "difference files" were scored by the researchers with reference to standard SSI signalling manuals. The results of this scoring procedure were reviewed by the senior signalling engineer who devised the test in the first place, and two changes were made to the scoring.

Faults were categorised according to the signalling principles which they violated and whether they were acts of omission or commission. This categorisation was broadly determined upon the basis of the HTA.

It was necessary to use different criteria when scoring the frequency of faults relating to the rule- vs. knowledge-based task components. With respect to the rule-based task component, faults were recorded upon the basis of the functional accuracy of the code (i.e., according to the correctness of individual functions). This avoided the problem of participants being penalised many times for one conceptual error. For instance, code must be written for every sub-route, specifying the conditions under which that sub-route is released, and this code may consist of many individual 'words'. However, some participants overlooked a whole sub-route, resulting in the omission of many items of code even though they had probably committed only one 'error'.

With respect to the knowledge-based component of the work sample, errors were recorded upon the basis of the accuracy of individual "words" within the completed code. This was due to the fact that this part of the code was short (four lines) and that each separate item fulfilled a specific function. That is, errors for each item of code would not be confounded with those for other items of code that were nearby in the file. This is not the case for the rule-based code, where, as discussed above, whole lines of code may be highly related i.e. tightly coupled (cf. Perrow, 1984).

Owing to their inexperience two participants were unable to attempt the knowledge-based component of this work sample; therefore their data were excluded from the analysis below, leaving $N = 13$.

All tests of statistical significance are two-tailed with $\alpha = .05$ unless otherwise stated.

Table 1 shows task completion times. There was a large variation in the time taken to complete the work sample (the fastest participant took 5,005 seconds and the slowest took 17,781 seconds [almost five hours]).

Table 1: Means and standard deviations for component and total task completion time (s).

Task component	Completion Time	
	M	*SD*
Rule-based performance	8,983	2,798
Knowledge-based performance	1,120	453
Total completion time	10,103	3,013

There was also a large difference in the time taken to complete the rule- vs. knowledge-based aspects of the task, with the rule-based performance taking on average more than eight times longer than the knowledge-based performance. However, if the amount of code written is taken into account, the rule-based work (120 lines) took on average 75 seconds per line, whereas the knowledge-based work (4 lines) took on average 280 seconds per line, or about four times as long per line.

Correlations between speed and accuracy were not significant, although there was a tendency for faster rule-based and knowledge-based performance to be associated with increased faults in these task components.

Table 2 presents descriptive statistics for fault performance. Again, there are sharp differences between performances at the rule- vs. knowledge-based levels. Rule-based work accounted for approximately twice the number of faults as the knowledge-based task. However, again taking into account the size of each section of work, rule-based performance (120 lines of code) led to 0.06 faults per line whereas knowledge-based performance (4 lines) led to 0.92 faults per line of code, or about 15 times the number of faults per line.

Correlations between speed and accuracy were not significant, although there was a tendency for faster rule-based and knowledge-based performance to be associated with increased faults in these task components.

Table 2: Means and standard deviations for component and total faults.

Task Component	Faults	
	M	*SD*
Rule-based performance	7.15	4.06
Knowledge-based performance	3.69	1.84
Total	10.85	4.36

3.1.2.1 Fault categorisation

Table 3 shows a breakdown of fault frequencies by signalling principle violated and fault type, i.e. whether the fault was one of omission or commission.

Table 3: Breakdown of faults by signalling principle violated for the data writing work sample

Signalling Principle	Faults		
	Omission	Commission	Total
Rule-based task component			
Identity and labelling	0	2	2
Route setting	17	14	31
Aspect control	25	6	31
Approach locking	7	26	33
Opposing locking	4	1	5
Aspect sequence	3	3	6
Other	21	13	34
Sub total	77	65	142
Knowledge-based task component			
Siding occupation latch	32	16	48
Total	109	81	190

Looking first at the rule-based performance data, the route-setting, aspect control, approach locking and 'other' categories accounted for the largest number of faults. Acts of omission and commission were roughly equally frequent, but acts of omission were more prevalent in conjunction with aspect control faults, and acts of commission were comparatively frequent with regard to approach locking faults.

Loglinear analysis of rule-based performance showed a significant association between signalling principle violated and fault frequency (χ^2_{lr} [6, N = 142] = 81.93, p < .0001). There was no main effect of fault type (omission /commission), but the interaction between signalling violation and fault type was significant (χ^2_{lr} [6, N = 142] = 30.03, p < .0001). This is attributable to the relatively large number of faults of omission for Aspect Control, and the large number of faults of commission for Approach locking.

For the knowledge-based component of the task faults of omission were more frequent than those of commission (χ^2 [1, N = 48] = 5.33, p < .05).

3.1.2.2 Common-mode error

It was found that a number of the faults committed in the work sample test were made by more than one participant, and all of these faults were found within the rule-based section of the task. (In the knowledge-based code every participant made an error of some sort, but the faults were not identical.) However, in the rule-based part of the task there were eight specific faults that were made in identical fashion by more than one participant: two which were made by three participants, two of which were made by four participants and a further four which were made by more than half of the sample. These latter four faults are considered to reflect common mode failure, as based upon these data they were more likely to be committed than not.

Although, as noted earlier, determining common faults does not necessarily mean that a common mode psychological error has been identified, it is strongly suggestive of a common process leading to error and this information nevertheless may be useful in identifying weaknesses in the design process.

The four "common-mode" errors are described below.

- All participants made an 'approach locking' fault relating to SSI timing routines. According to the signalling manual a track circuit should have been occupied by a train for more than five seconds in order to prove that the train was definitely in the particular section of track. However, due to SSI timing limitations it is necessary to allow a two second margin for timing error, if this is a significant proportion of the timed interval. Consequently, the timing check in the data should have been for a period of seven seconds. All participants put down a timing period of five seconds.
- Eleven participants made an 'aspect control' fault concerning unconditional lamp proving. Most signals have an extra circuit that shows the signal controller if it is alight, and the SSI programme will only use the signal conditional upon this circuit being operational. Some non-critical signals, however, do not have the extra circuit and must be set to light "unconditionally". The 11 participants instead wrote the default code.
- Nine participants made an 'aspect control' fault relating to a siding which required last wheel replacement. Usually, a signal should turn red after the first axle of the train has passed it. However, for trains shunting into a siding backwards this would result in the driver (now at the back of the train) being shown a red light while still moving past the signal. For siding signals, then, the signal should turn red after the last wheel of the train has passed. These participants failed to include the necessary code to achieve this.
- Seven participants made an 'other' fault relating to clearing the automatic working function of a signal. Usually, signals are set to allow operation by automatic timetabling software as well as signallers, but sometimes (as here) this functioning must be turned off. Code which should have been included in order to cancel the automatic working of the signal was missed out.

There was no significant association between the occurrence of these faults and the data collection site, i.e. the faults did not appear to be site or company specific.

These common-mode errors all seemed to represent situations where habitual behaviour (i.e. writing default, standard code) occurred instead of the more appropriate but less common behaviour. This has been termed variously "Einstellung" ("mind-set"; Luchins, 1950) and "strong-but-wrong" habit intrusions (e.g. Norman, 1988; Reason 1990).

3.1.2.3 The effects of cognitive ability

Table 4 presents descriptive statistics for measures of cognitive ability, along with normative data collected from a stratified sample of 4000 of the U.S. working population at the time of the development of these tests (US Department of Labor, 1982).

Table 4: Means and standard deviations for cognitive ability: Work sample and normative sample.

Cognitive Ability	Work sample (n = 13)		Normative sample (n = 4000)	
	Mean	**SD**	**Mean**	**SD**
Clerical perception	60.23	10.60	46.68	17.89
3-D spatial ability	25.54	6.45	15.80	6.10
Vocabulary	29.92	7.26	20.14	10.23
Tool matching	29.46	4.59	30.72	7.41
Arithmetic reasoning	13.92	2.63	11.02	4.24

As can be seen, the present sample is of comparatively high spatial ability and clerical perception ability. In addition, the mean aptitude score on a measure of fluid intelligence (derived from a composite of scores upon the three-dimensional spatial ability, vocabulary, and arithmetic reasoning tests) was 123.33 (SD = 13.99) indicating that the sample was also of comparatively high fluid intelligence (normative parameters: $\mu = 100$; $\sigma = 20$)

A series of correlations were used to examine the effects of cognitive ability in relation to data preparation performance. The only significant correlation was between three-dimensional spatial ability scores and completion times for the knowledge-based task component ($r[13] = -.65$, $p < .05$), with high spatial ability individuals performing more quickly than low spatial ability individuals. There was a non-significant tendency for fluid intelligence to be associated with both quicker ($r[13] = -.38$) and more accurate ($r[13] = -.49$) performance upon the knowledge-based task component.

3.1.2.4 The effects of personality

Table 5 presents the STEN scores for each of the main PREVUE personality test scales. As can be seen, the present sample was within the central range for each of these factors, although the mean for the Extraversion scale was significantly less than the reference population ($t[13] = 2.93$, $p < .05$). The effect of each of the main and minor factors was considered in relation to performance upon the work sample

Table 5: STEN scores for PREVUE scales

Personality Scale	STEN	
	Mean	SD
Independence	5.69	1.75
Conscientiousness	5.54	1.76
Extraversion	3.92	1.63
Stability	4.92	1.55

There was significant correlation between the minor scale I1 (tough minded, competitive) and rule-based completion time ($r [13] = -.53$, $p < .05$), and with total completion time ($r [13] = -.57$, $p < .05$), although the trend for I2 (forthright, assertive) was in the opposite (positive) direction. The correlation between stability (major factor) and total number of errors just failed to reach significance ($r [13] = -.55$, $p = .051$). However, this association was significant ($r [13] = -.65$, $p < .05$) for S1 (unruffled, not easy to upset or annoy). S1 was also significantly correlated with knowledge-based errors ($r [13] = -.56$, $p < .05$). In all these cases, high stability was associated with less error-prone performance. It may be that these results can in some degree be attributed

to the experimental situation, with competitive individuals performing more quickly, and stable, unruffled individuals performing more accurately. However, there is some previous research evidence to suggest that stability may be associated with reduced errors when performing an inspection task (Hsu & Chan, 1995).

E1 (sociable, outgoing) was significantly correlated with knowledge-based performance ($r\ [13] = .56$, $p < .05$), such that high extraversion was associated with slower performance. Extraversion (combined) was also associated with knowledge-based errors ($r\ [13] = .70$, $p < .01$) with high extraversion being associated with more error-prone performance. Once again this appeared to be primarily attributable to E1 ($r\ [13] = .81$, $p < .001$), rather than E2 (group dependent). This result is consistent with previous research which has examined personality differences in relation to computer programming tasks, in which introversion has also been associated with better performance (see Westerman, 1993, for a review).

3.1.2.5 The effects of mood

A series of t-tests revealed no significant differences in pre- vs. post-task mood measures. Mood scores were therefore taken as an average of these measures.

There was a significant association between general arousal and rule-based completion time ($r\ [13] = .54$, $p < .05$), and the association with knowledge-based completion time just failed to reach significance for both general arousal ($r\ [13] = .47$, $p > .05$) and energetic arousal ($r\ [13] = .48$, $p > .05$). Surprisingly, in all cases the nature of this association was such that high arousal was associated with slower performance. It may be that these correlations are indicative of an association between arousal and conscientious task performance, such that participants who reported high energetic arousal took longer in an effort to minimise errors. However, the only significant correlation with the number of errors made was between tense arousal and total errors, such that high tense arousal was associated with fewer errors ($r\ [13] = -.59$, $p < .05$). Further analysis suggested that this effect is attributable to pre-task tense arousal ($r\ [13] = -.66$, $p < .05$) as opposed to post-task tense arousal ($r\ [13] = -.20$, $p > .05$).

3.1.2.6 The effects of experience

There was a non-significant trend for the amount of SSI data preparation experience to be associated with rule-based ($r [15] = -.50$, $p = .058$), knowledge-based ($r [13] = -.48$, $p = .092$), and total ($r [13] = -.35$, $p > .10$) completion times. Data preparation experience was significantly associated with rule-based ($r [15] = -.67$, $p < .01$), and total ($r [15] = -.60$, $p < .05$) errors, with participants with greater experience committing fewer faults.

3.1.3 Summary

In most respects the results from the work sample test were in agreement with previous research. Looking at individual differences first, good performance (quick, accurate) was associated with greater experience, higher spatial ability and greater emotional stability. Given the restricted ranges of many of the variables and the small sample size of the study these results are encouraging if unsurprising.

To a certain extent the results are similarly predictable for the differences between rule- and knowledge-based performance. Knowledge-based performance is both slower and more error prone than is rule-based performance, as is commonly found (e.g. Reason, 1990).

However, it was the opposite state of affairs when looking at common-mode error. Although errors are relatively less likely in rule-based vs. knowledge-based performance, when they do occur they are likely to manifest themselves in a much smaller set of possible faults than would be likely for knowledge-based errors. In addition, the faults generated during rule-based performance are also more likely to resemble correct performance than those at the knowledge-based level. In short, it seems that although rule-based errors are overall less likely to occur, when they do occur they are more likely to look like instances of correct performance than are knowledge-based errors.

This has major implications for the way in which common-mode errors are detected. In domains such as statistical problem solving (Allwood, 1984) and computer database

use (Rizzo, Bagnara &Visciola, 1987) it has been found that participants engaged in three main types of behaviour when detecting and correcting their errors:

- *Direct error hypothesis* behaviour. This is behaviour directly focused on a real or suspected fault of known type and location.

- *Error suspicious* behaviour. Although no actual or suspected fault had been identified, procedures adopted did not lead to the expected conclusion.

- *Standard check* behaviour. In contrast to the above two behaviours, standard checks are not dependent on task feedback, and merely reflect the good practice of reviewing and checking work as it progresses.

Direct error hypothesis behaviour has been found most often in skill-based slips and lapses, where the quality of feedback from an action is good, often showing immediately that an error has been made. Error suspicious behaviour is observed more in the detection of Rule- and Knowledge-based mistakes, where the coupling of errors and outcomes is less immediate and potent. However, in the case of the work sample, because the common-mode errors were so similar to contextually appropriate correct performance, and because feedback from the task was poor, error suspicious behaviour was not triggered. It is not clear from these data whether participants carried out standard checks and failed to detect these errors, or whether they simply did not carry out standard checks (perhaps because of the time pressure imposed by the work sample situation). However, it seems unlikely that standard check behaviour would routinely detect the highly situationally appropriate faults such as the common mode errors observed in the work-sample test.

The work sample test has shown that intelligent, experienced and motivated engineers are, like anyone else, prone to error. Some of the faults observed seem attributable to lack of knowledge or simple slips, say, and could be expected to be readily 'visible'. However, some of the faults do not seem to be the product of idiosyncratic factors but instead seem to be predictable based upon the status of the task as "exception handling". Based upon their prevalence it seems unlikely that these errors would be

quite so 'visible' in the data writing stage of the DPP and their eradication would therefore depend upon the veracity of subsequent task stages. These will be evaluated next.

3.2 Error / fault logs

Currently, as a routine part of the DPP, data on faults found at the checking and testing stages (goals 1.4 and 1.5) are produced in the form of 'error' logs. This section presents an analysis of all the fault data contained in these logs which were made available by the participating organisations. These logs will from this point on be referred to as "fault logs" to preserve the usual distinction between an error (a psychological process) and a fault (an observable flaw in a system).

The main purpose of the fault logs is to communicate the presence of known or suspected faults in the SSI code from the checker or tester to the writer, so that the SSI code can be fixed. To preserve the independence of these stages of the DPP the information in the logs is necessarily brief. The writer is given broad details of the fault, such as its location in the code or its manifestation in testing behaviour, but no guidance is given as to the necessary remedial actions. Although this practice is valuable within the immediate context of the DPP, providing some degree of safeguard against the propagation of common-mode error, it means that the amount of information available for analysis is limited.

Furthermore, there is much between-organisation (and even between-site) variation in the form which these fault logs take, particularly with respect to checking logs. An additional caveat is that the fault logs refer only to problems found in the office-based DPP, not to on-site testing and post commissioning errors (if any). Unfortunately, no data of this nature were made available. That said, the fault logs do document actual errors made during the DPP, and therefore have intrinsic validity.

3.2.1 Method

Fault logs were gathered from seven organisations (including sub-contractors), across nine sites and relate to 12 different SSIs. Faults were initially categorised by two raters working together according to the signalling principles violated (as for the work sample). A small number of the fault log categorisations (about 5%) were checked with

professional SSI engineers and revealed no instances of disagreement over the error categorisation.

Limited data were available with respect to the effects of scheme complexity, and the iterative design process, i.e. how many cycles of writing, checking and testing the SSI geographical data had been through.

It should be noted that there are a number of reasons why it is not possible to use these data to make quantitative comparisons of the relative efficacy of the checking or testing stages of the DPP. First, as mentioned, fault data were gathered from a number of different schemes and the checking and testing data are not completely matched. Second, if faults are detected at the checking stage it will obviously not be possible to detect them at the testing stage, and consequently there is no means of estimating the efficacy of the testing process in detecting them. Third, there are no data relating to the number of faults that were initially present in the data. Finally, there were no data available relating to faults detected at, or following, the 'on-site testing' phase of the DPP. Once again, this makes the appraisal of the efficiency of the testing stage problematic. Nevertheless, given these qualifications, a qualitative appraisal of the relative error frequencies nevertheless provides useful information on factors possibly affecting the reliability of the DPP.

3.2.2 Results and discussion

A breakdown of 1021 faults logged at the checking or testing stages of the DPP by signalling principle contravened is presented in Table 6.

The precise meaning of the various signalling principle categories in Table 6 will not be discussed much here, because to do so would require a significant detour into railway signalling lore (for those really interested, Hall, 1992 provides a good introduction). For the purposes of this thesis the various categories can be thought of as similar to various categories of law-breaking, perhaps, for which two systems of detection (checking and testing) are being compared. The law-breaking example might bring to mind an objection to this approach, because different types of lawlessness vary in their 'seriousness', and so surely different faults categories vary in their importance too.

However, as argued by Broomfield and Chung (1995), there is no established technique for mapping software faults on to system hazards anyway. (Stated a good few years earlier by Benjamin Franklin (in1757); "A little neglect may breed great mischief; for want of a nail, the shoe was lost, for want of a shoe the horse was lost...".)

Table 6: Number of faults logged during checking and testing by signalling principle contravened.

Signalling Principle	Checking	Testing	Log OR
Identity and labelling errors	44	36	.00
Route setting	104	150	-.70
Signal aspect control	48	76	-.66
Approach locking	32	41	-.44
Opposing locking	27	64	-1.13
Aspect sequence	16	11	.42
Other	175	32	1.83
Sub total	446	410	-.45
None (no fault, false alarm)	102	63	.45
Total	548	473	

Note: OR = odds ratio

So, the signalling principles violated will have for now to serve merely as a mechanism for comparing the relative efficacy of checking and testing fault detection performance. To aid interpretation the log odds ratio of faults detected by checking and testing in each category is given. This provides a symmetrical index of the relative efficacy with which checking and testing detect faults of the various types. The more positive the log odds ratio, the greater the number of faults detected at the checking stage in relation to the number detected at the testing stage, and vice versa for negative values.

Identity and labelling faults (the misnaming of data and screen objects) had a log OR of zero, indicating parity in performance between checking and testing. However, it

should be noted that in the case of testing these faults relate solely to the simulation screen (i.e. all other naming faults appear to be detected at the checking stage).

Route setting was the category which accounted for the greatest percentage of faults, and like all of the other categories bar 'aspect sequence' and 'other', the log OR was negative, indicating superior fault detection by testing.

The difference between checking and testing performance was greatest for faults in the 'other' category. These are faults which did not obviously fall into any of the previously defined categories. Problems in the layout of the SSI code accounted for the biggest single sub-category of 'other' faults detected during checking (23.43%). Although this type of fault, which cannot be detected during testing, is not considered a safety-critical feature of the code, it is likely nonetheless to contribute to the efficiency of the design process, determining the speed and accuracy with which data can be checked, and the ease with which subsequent reworking can be done.

With respect to 'other' faults detected at testing, the highest percentage was accounted for by control-table flaws (43.75%). Although these data suggest that these faults are not easily detected during the checking phase of the DPP it should be noted that the faults included in this analysis relate only to instances where the control tables were demonstrated to have been in error, which is most easily done during testing. There were also an additional 36 faults reported at the checking stage (not included in the present analysis) for which there was no evidence available to indicate whether the control tables were in fact inaccurate. Nevertheless, it may be that testers have a better overview of the functionality of the system by virtue of training, experience, or task environment.

The fault category in which testing outperformed checking by the greatest margin was in 'opposing locking'. A further breakdown of these data revealed that, of those fault reports where sufficient detail was available to make an assessment, an important factor was whether the opposing locking code involved the use of code-constructs called 'sub-routes'. Sub-routes are used to specify the direction a train will take over a section of track, which is important when deciding if a set of points needs to be locked

in place if a train is about to travel over them (the train will derail if the points are moved under it). The difficulty in working with sub-routes will be addressed in § 5.

A high percentage (16%) of all faults logged were false alarms (i.e., a fault was reported where in fact none existed). Although there are undoubtedly safety-related advantages associated with the use of a lax response criterion (the preparedness of the checker or tester to signal a problem), these false alarms will inevitably reduce the efficiency of the DPP because the data preparer must prepare a response to each fault logged. A further breakdown of these false alarms revealed that, of those occurring during checking, a high percentage resulted from either: a) functions which were required but dealt with elsewhere in the code (12.7%), or; b) particular scheme-specific requirements (23.53%). Of those false-alarms occurring during testing, scheme specific requirements also accounted for the largest percentage of false alarms (20.63%). "Scheme specific requirements" is another way of saying that these are instances of exceptions to the normal specifications.

3.2.2.1 The iterative process of fault detection

Limited amounts of data were available relating to the stages at which faults were logged during the developmental life-cycle of individual SSIs. Table 7 presents data relating to a single scheme for three consecutive data checks followed by a test. (There were four checks, but by definition the final one detected no faults.)

As can be seen, by far the largest proportion of faults (84.40%) was detected during the first check. One fault in the 'other' category, relating to the layout of the code, was detected at the first check, remained uncorrected, and was detected again at the second check. All of the other faults found on the first and subsequent checks were fixed following their first report, therefore 16 unique faults 'survived' the initial check, 12 passed the second check, and 10 faults made it all the way through the checking process (i.e. those 10 detected at testing). Of note is the single 'opposing-locking' fault that survived the first check. The overall number of 'opposing-locking' faults was very low, only three in total, yet one of them passed initially undetected. Although only one datum, this fits with the earlier finding, that opposing-locking code is difficult to check.

Table 7: Number of faults logged across three checking and one testing stage by signalling principle contravened

Signalling Principle	Checking cycle			Testing
	1	2	3	
Identity and labelling errors	15	2	0	3
Route setting (general)	9	0	0	0
Signal aspect control	6	0	0	1
Approach locking	5	0	0	0
Opposing locking	2	1	0	0
Aspect sequence	3	0	0	1
Other	52	2	2	5
Sub total	92	5	2	10
None (no fault, false alarm)	2	0	0	6
Total	94	5	2	16

Table 8: Number of faults logged across four checking stages by signalling principle contravened

Signalling Principle	**Checking cycle**			
	1	**2**	**3**	**4**
Identity and labelling errors	0	0	2	3
Route setting (general)	22	1	2	0
Signal aspect control	2	0	1	0
Approach locking	1	0	1	0
Opposing locking	3	0	0	0
Aspect sequence	1	0	1	0
Other	24	2	1	4
Sub total	53	3	8	7
None (no fault, false alarm)	17	4	4	0
Total	70	7	12	7

Table 8 presents the data relating to four sequential data checks on the same code. Once again a large proportion of faults (74.65%) was detected during the first check. Of the faults detected at later checks, identity and labelling faults accounted for 27.78%. Non-standard requirements also accounted for two faults detected at the third check. Of the 'other' faults detected at the fourth check, three of these related to 'cosmetic data changes'.

3.2.2.2 Scheme complexity

In order to examine the effects of work complexity on fault detection, the faults detected at the first check and first test were examined for SSI schemes comprising less than 30 routes (three schemes with 14, 17, and 19 routes, respectively), and more than 30 routes (three schemes with 32, 33, and 75 routes). As above, only a brief qualitative examination of the data is presented, as data relating to checking and testing do not necessarily relate to the same schemes. Of particular concern in this respect is the small quantity of testing data for simple schemes.

Table 9: Number of faults logged at the checking and testing stages of the data preparation process for schemes with less than 30 routes.

Signalling Principle	Design Stage		Total
	Checking	Testing	
Identity and labelling	23	3	26
Route setting (general)	57	1	58
Signal aspect control	23	4	27
Approach locking	14	3	17
Opposing locking	19	0	19
Aspect sequence	11	0	11
Other	106	8	114
Sub total	253	19	272
None (no fault, false alarm)	19	6	25
Total	272	25	299

As can be seen from Table 9 and Table 10, the greatest percentage of faults detected for simple schemes were 'other' faults (41.91%), whereas 'route setting' faults accounted for the greatest percentage of faults detected in the more complex schemes (39.31%) and 'other 'faults accounted for only 9.20%.

Table 10: Number of faults logged at the checking and testing stages of the data preparation process for schemes with more than 30 routes

Signalling Principle	Design Stage		Total
	Checking	Testing	
Identity and labelling	4	33	37
Route setting (general)	24	147	171
Signal aspect control	4	67	71
Approach locking	1	36	37
Opposing locking	3	64	67
Aspect sequence	2	10	12
Other	23	17	40
Sub total	61	374	435
None (no fault, false alarm)	43	52	95
Total	104	426	530

False alarms accounted for only 8.36% of those faults reported for simple schemes, whereas this percentage was 17.92% for the more complex schemes. It is possible that this is attributable to design engineers adopting a more lax response criterion as complexity increases.

An examination of the proportion of faults detected at the checking vs. testing stages of the DPP for schemes of differing complexity suggest that checking is less efficient at detecting all types of faults when scheme complexity is increased. However, it is worth noting that the checking process apparently detected all 'opposing locking' and 'aspect sequence' faults in simple schemes.

3.3 Video recording task performance

The preceding analyses were concerned with identifying faults generated by a data writer which remained undetected at that time, requiring detection at a subsequent stage of the DPP. However, it is highly likely that during each of the design phases a number of errors (primarily slips) occur which are generally detected at that time (and are therefore not recorded) but which decrease the efficiency of the design process and are likely to be indicators of task difficulty. It may be that errors which are apparent with relatively low frequency when checking or testing a previous phase of the DPP, in fact occur with high frequency, but the majority of these occurrences are 'self-detected' (e.g. Rabbitt, 1978).

Investigating these errors, particularly within the context of such a complex task, is difficult. The use of concurrent verbal protocols was considered as a possible method (see Ericsson and Simon, 1984). This technique requires that participants give a verbal commentary while performing their task. The advantages of this include: high face validity; ease of use in applied settings with minimal disruption, and; some access to the cognitive processes associated with task performance, which would not otherwise be readily accessible.

However, there are also a number of associated drawbacks. With certain types of cognitive performance (e.g. automatic processing or processing spatially encoded material) it may be that the mechanisms underlying performance are not readily available for conscious verbal report (Yang, 2003) Crucially, however, there is evidence to indicate that task performance whilst giving a concurrent verbal protocol may be faster (Berry and Broadbent, 1990) or less error prone (Wright and Converse, 1992) than normal task performance. This is obviously unacceptable in the context of this research.

For this reason a variation on Rabbitt's (1978) "caught in the act" or "oops" technique was used, coupled with elements from "video walkthroughs" (Diaper, 1989), described next.

3.3.1 Method

Task performance was video recorded in a similar manner to that described for the work sample (§ 3.1.1.3), with the exception of testing where one camera was focused upon the signalman's screen and the other was focused upon the trackside screen. The observation comprised seven and a half hours of writing, three hours of checking, six and a quarter hours of testing, and one and a half hours of simulation screen design (strictly, a part of the set-to-run task, but here included for its similarity to testing). Observations were all carried out at Westinghouse Signals' Chippenham site. Two participants were observed for date writing, three for testing, and one participant each for checking and simulation screen design.

Task performance took place in as natural a manner as possible, with participants only required to signal when they had detected that they had made, or had just stopped themselves making, an error (Rabbit, 1978). Depending upon the complexity of the error, a brief description was given by the engineer at the time, or an explanation given following task performance whilst viewing a replay of the video recording (Diaper, 1989). To try to reduce the likelihood of faking a lack of errors the observed participants were informed that their recording would be reviewed by another engineer at a later date (although this did not, in fact, occur).

3.3.2 Results and discussion

3.3.2.1 Writing

In total there were 30 errors signalled by the participants over a combined period of seven and a half hours of writing, or four per hour. Of these 17 were typographical errors and two related to wrong file locations. Of note were four slips which were identified as relating to opposing locking, three of which related to specifying sub-routes.

3.3.2.2 Checking

The checking phase of the DPP was analysed for a period of three hours. However, no errors were recorded during this time.

3.3.2.3 Testing

Errors identified at the data testing phase of the DPP included: incorrect identification of 'stations' (areas) because of the scrolling requirements of the interface and poor labelling methods (this occurred twice), and; selection of the wrong route to perform a test upon. In total five errors were identified during six and a quarter hours of testing, just under one per hour.

3.3.2.4 Simulation screen design

In one and a half hours of observing the process of simulation screen design three errors were recorded (two per hour), all of which were typographical errors.

3.3.3 Summary

Overall, few errors were self-detected by the SSI engineers as they performed their various tasks. Viewed very broadly, the number of self-detected errors seemed to vary with the amount and type of manual control input required of the operator. Listing in descending order of control input (and self-detected error rate): writing and simulation screen design both involve mouse and keyboard work; testing involves trackball use; while checking does not require any substantive control actions to be performed (turning a page, etc.).

It may be that this observational method of analysis is only useful in recording errors in relation to overt actions, and not covert cognitive processes. "There is some evidence...that while people are good at catching their own errors of action, they are much less good at catching their own errors of thinking, decision making, and perception" (Senders and Moray, 1991, p.78).

3.4 Existing documentation and semi-structured interviews

As a means of validating the data reported above, and also to gather information relating to rare errors, additional information was gathered from existing documentation and the semi-structured interviews used for the HTA. The following are the main points to arise.

3.4.1 Writing and/or checking errors

A number of specific errors, or potential sources of error, were identified as being common to the processes of writing and/or checking data. These included:

- The copying of information from one section of a file to another without making all the necessary alterations required for the new context.
- Insufficient information contained in comments making code hard to comprehend.
- Data being laid out so that it is difficult to check the syntax (e.g. indicating more than one blank line by putting several full stop, line terminators, on one line).
- Misinterpretation of sub-routs.
- Code duplication (which may lead to timing errors).
- Interfaces to other, non-SSI, equipment

Further, at a more general level, it was reported that it is easier to check existing code for correctness rather than to determine whether necessary code are missing; and that writing errors tend to spring from unusual scheme requirements.

3.4.2 Testing errors

As stated earlier, no little evidence relating to faults detected at the on-site testing stage, that would reveal errors at the DPP testing stage, was made available. However, anecdotal evidence was obtained in interviews to indicate that faults do get past both the simulation and on-site testing stages, i.e. into service.

Generally it would seem that faults which are not detected by the simulation testing are concerned with:

- Cross boundaries functioning (where the SSI system in control of a section of track must communicate with a neighbouring interlocking)
- The interface with trackside equipment

- Timing errors (where processing is not completed by the SSI modules in the allotted time)
- Multiple route-setting.

Cross boundary faults may arise because of poor communication between design teams (incorrectly identifying the allocation of functions to each other, for instance).

Some of the faults that are detected on-site, for instance certain trackside equipment and multiple route setting faults, cannot be replicated by the SSI simulator and therefore it is no surprise that these occasionally make it through to on-site testing. However, on occasion basic code faults are reported to come to light at this stage.

Finally, faults which have been detected in commissioned schemes (after on-site testing) tend to arise from either very complex code, where the fault is only 'visible' when multiple interdependent conditions are fulfilled, or cross- boundary conditions involving communication with non-SSI equipment.

3.5 Conclusions

In drawing conclusions upon the basis of the data reported above, some qualifications should be noted. First, the number of participants who could be recruited to the work sample was necessarily limited and conclusions drawn on the basis of the correlational statistics must be tentative. Second, the fault log data are all that were available at the time, and the collection of additional data, particularly from each iterative phase of the DPP, would allow more confidence in conclusions. Finally, as mentioned above, only limited information relating to faults which are detected at the 'on-site' testing phase, or following commissioning, was available. If the reliability of the testing process is to be properly evaluated then these data are essential. Without it, the conclusions that can be drawn relating to the reliability of the testing process are somewhat limited.

3.5.1 Diversity in the Design Process

There are many beneficial features of the DPP that seem to contribute to its reliability. The present arrangement of checking and testing produces a good deal of task diversity, as supported by several instances of different types of error being detected at

different stages of the design process. Although these errors were categorised earlier by signalling principle, it was also noted that it seems that checking is particularly good at detecting non-safety-critical function errors which relate to the 'liveness' of the railway and the readability (reusability) of code. By comparison, testing is good at detecting errors which relate primarily to the safety of the railway.

In contrast to task environment diversity, personnel diversity seemed to show the opposite trend, at least when it came to the writing task. In the writing work sample task a number of faults were made identically by experience and inexperienced staff from various different companies. In such instances of 'common mode' failure, errors will be resistant detection within the DPP because design engineers, perhaps because of similar background, training, or ability, approach the task in a similar manner. It has been argued elsewhere (e.g. Westerman, Shryane, Crawshaw, Hockey & Wyatt-Millington, 1995) that one solution to common mode human errors lies in cognitive diversity, such that individuals approach a checking or testing task utilising different cognitive models/strategies. The investigation of diversity in the SSI DPP will form the bulk of the remainder of this thesis, from § 6 onwards.

3.5.2 Complexity

A number of broad factors can be identified as making faults particularly resistant to detection during the DPP. As might be predicted, the complexity of the code appears to be associated with the propensity for human error. If the functionality required from the SSI is complex, this makes faults more difficult to detect. This may be attributed to human attentional limitations which result in mistakes of 'bounded rationality' and 'imperfect rationality' (see § 1.3.1).

In order to reduce these effects it is necessary to alter the ratio of attentional demands (the requirements of the task) to attentional resources (the capacities the design engineer brings to the task). This might be achieved by: a) recruiting design engineers who have greater cognitive abilities (increased attentional resource supply); b) employing training strategies, such that task performance requires fewer attentional resources (see Anderson, 1993), or; c) redesigning the task environment such that there is a reduced information processing load placed upon the design engineer.

There are a number of ways in which redesigning the task environment could reduce attentional demands. First, by incorporating more task support information in the task environment (e.g. display based reasoning; Howes and Payne, 1990). This could take the form of checklists of sequences of required actions, or potential errors (e.g., see Layton & Johnson, 1993); or the use of multiple windows, facilitating the performance of tasks which require cross checking of items from a number of different sources (see Miyata and Norman, 1986). In order to gain some of the benefits associated with these techniques it may be worthwhile making the checking phase of the DPP more of a computer-based process than is currently the case.

Second, there are changes which can be made to the task environment which serve to make errors more 'visible' (see Norman, 1988). In order to facilitate this process computer-based display techniques may be used, such as the three-dimensional representation of data functions or railway layouts, which assist the design engineer in 'visualising' the interplay of variables in complex schemes.

However, the importance of 'context' in this respect cannot be overstated. A model that has been successfully used to provide increased contextual information (see Vicente and Rasmussen, 1992) is the Abstraction Hierarchy (Rasmussen, 1986). This model is based upon the premise that any given engineering system (task environment) can be described in terms of a number of different levels of abstraction. Lower levels of the hierarchy are concerned with how certain system functions are implemented (the 'nuts and bolts' of the system), whereas higher levels are concerned with why system functions are required. Within the SSI DPP, additional contextual support could be provided at the checking stage in the form of 'why' information; and at the testing stage in the form of 'how' information. It should be noted, however, that this approach may result in the checking and testing tasks becoming less diverse, and that although more errors may be detected as a result of these changes the potential to avoid common mode errors may be reduced.

Novel or unusual requirements also appear to be an important factor in promoting error and making errors more resistant to detection. There is evidence to suggest that this may in part be attributable to design engineers making 'mistakes of reluctant

rationality'; i.e., employing familiar performance strategies rather than engaging in the increased cognitive demand associated with calculating novel solutions (e.g. Reason, 1990). With this in mind, it is worth noting that a high percentage of the false alarms recorded at both checking and testing phases of the DPP were attributable to scheme specific requirements.

However, undoubtedly novel data requirements will also result in 'mistakes of imperfect rationality', such that the design engineer has an imperfect mental model of the task environment. An important improvement which could be implemented in this area concerns the 'usability' of support documentation. For example, the accessibility of documentation, the ease of search for target information, and the ease of updating documentation could all be improved by providing on-line, i.e. computer-based, documentation (Layton and Johnson, 1993). Further changes to reduce errors resulting from unusual data requirements may also include an increased use of checklists, enforced cross-checks with increased functional redundancy, and the use of inspection teams (see Fagan, 1976; 1986). Each of these techniques serves to test the assumptions which are being applied by the design engineers to the task at hand. It can also be argued that improvements in the ratio of attentional demands to attentional resource supply, as described in the previous paragraph, will facilitate novel problem solving (see Ohlsson, 1984a, 1984b, 1985). Consequently, techniques such as data visualisation can also make an important contribution to the reduction of these types of errors.

Finally, limitations in the simulation equipment increase the difficulty of the testing task, and make it impossible to test for some complex errors. Although some of the changes which could usefully be implemented in this area are beyond the scope of the present investigation, a number of human factors issues can be identified. For example, scrolling requirements and labelling conventions have been found to cause orientation problems during testing (see § 3.3.2.3). A combination of improved equipment specifications and the application of basic human factors principles could result in improvements in this respect.

It is worthy of note that checking seemed to be particularly poor at detecting opposing locking faults, particularly those associated with the use of 'sub-routes'. The

observation analysis reported above (see § 3.3) suggests that slips are frequent when writing opposing locking data, although given the relative frequency of these faults it would appear that most are self-detected.

3.5.3 Automatic Data Preparation

Given the type of errors which are resistant to the DPP process, i.e. errors which relate to complex and/or unusual data, it seems unlikely that Automatic Data Preparation (ADP) will have a substantial positive effect upon (i.e. reduce) the probabilities of these errors occurring. Automation is most easily applied to skill-based and rule-based task performance (see chapter 1). In contrast, those faults which are difficult to detect tend to be knowledge-based errors. Nevertheless, there are, skill-, and rule-based task components which humans apparently perform poorly, e.g., opposing locking errors during checking, to which ADP might usefully be directed. The benefits of such a strategy would be apparent in improved efficiency within the design process rather than improved overall reliability.

In § 2.2.4.2 the potentially damaging effects of the current move towards Automatic Data Preparation upon the training opportunities afforded design engineers was identified. The current strategy appears to be one of automating all that is technically possible. The danger of applying this strategy too rigorously is that it results in design engineers being left to complete only those (knowledge-based) task components which cannot be automated, and this may be "... a fragmented, difficult-to-perform job for which training is also a problem" (see Lockhart, Strub, Hawley, and Tapia, 1993, p. 1212). A viable alternative method of providing training would be to use Computer-Based Training. However, it can also be argued that less than maximum automation or the flexible use of automation may be useful alternatives which would allow design engineers to acquire the necessary skills within a meaningful framework.

3.5.4 Training

There are a number of areas of the DPP which can be identified upon the basis of these results as potentially benefiting from training interventions. First, several common mode errors were identified by the work sample. It would appear that these relate to deficiencies in the application of relatively straightforward principles (stated in

SSI 8003) which are independent of organisation and location. Second, training methods may be applied to develop skills appropriate to dealing with novel requirements (although see Patrick, 1992 for a discussion of some of the difficulties inherent in this type of endeavour). It is interesting to note that, in the work sample, DPP experience was associated with rule-based errors but not knowledge-based errors, suggesting that experience contributes to the acquisition of basic skills but does not necessarily impart the required skills for performing the more unusual task components.

3.5.5 Personnel selection

Comparison with normative data suggests that the current selection process favours engineers who are comparatively high in spatial ability, clerical perception, and fluid intelligence. The validity of spatial ability in this regard was supported by the negative association with completion times for the knowledge-based component of the work sample. The correlations between fluid intelligence and knowledge-based task performance, although not significant, were of a magnitude which suggests that they may also prove useful in the selection process. The fact that knowledge-based task components are the least amenable to automation highlights the future importance of efficient personnel selection in this regard.

Further to this, there was some evidence of an association between personality and DPP performance. Although some of the reported effects may have been attributable to the experimental situation, the association between extraversion and performance is consistent with previous investigations of computer programming, with introverts performing more quickly and more accurately. Similarly, the association between stability and performance, with stable individuals making fewer errors, is consistent with previous research concerning the performance of an inspection task (see § 3.1.2.3). Given that the sample were very much within the normal range of scores, it would seem that the selection process is not tapping these differences, and that personality testing could make a useful contribution in this area.

3.5.6 Sub-routes

The use of code constructs called sub-routes was repeatedly identified as a particular area of task difficulty. It is likely that this contributes to the difficulties encountered at the checking phase of the DPP in detecting opposing locking errors, as well as many other error types. Sub-routes will be investigated more thoroughly in § 5.

4 Mid thesis summary

The previous chapters were exploratory and primarily qualitative in nature. They also tended to focus on context-specific (i.e. SSI) issues rather than generally applicable findings.

What follows is a brief discussion of the main themes that emerged from the fieldwork, and how these themes will be operationalised and submitted to test in the latter half of the thesis.

4.1 Task analysis

The HTA described a system of office-based software design with three stages: one of production (writing) and two of quality control (checking and testing). It was observed that there existed diversity in the structure of the tasks that made up the quality control phases. One of the quality control tasks, checking, involved essentially the same representations of the same tasks that made up the production phase (writing). Both checking and writing involve the use of an abstract, text-based programming language and both dealing with SSI functioning at the 'micro' level, manipulating verbal and textual symbols, dealing with identical, abstract representations of the railway, i.e. in terms of bits, bytes and variables names (cf. Rasmussen & Lind, 1981).

The other quality control phase, testing, was markedly different. The representation of the tasks and the tasks themselves were distinct from those carried out in writing and checking, and were based more on concrete railway concepts and a visual-spatial task environment. In contrast to the writing and checking stages, testing requires the engineer to map the specifications directly on to the functioning of the railway network, as represented on the Graphical User Interface of the testing simulator. The tester deals with the 'macro', overall, functioning of the SSI, using a concrete, spatial representation of the railway (i.e. signals, points and tracks). It was proposed that these differences between checking and testing, and the associated differences in the demands they make on human performance, might manifest themselves in terms of the quantitative and qualitative aspects of patterns of faults detected at each stage.

4.2 Error Analysis

The error analysis consisted of a work sample test, task observation, and an audit of the fault logs used to record faults found by checkers and testers.

The qualitative differences found between checking and testing tasks in the task analysis were empirically investigated in the error analysis. For the fault log audit it was found that different fault types were differentially detected by the checking and testing stages. Simpler faults, and those affecting the basic functioning of the railway, were detected more frequently by checkers. More complex faults, and those to do with safety-related operations, were found more frequently in testing. This 'task diversity' may hold promise as a way to combat 'common mode' errors, such as those found in the work sample test.

The work sample test showed the influence of expertise and complexity on the ease with which fault-free task performance could be achieved. Routine work that could be completed with skill- and rule-based performance was far less error prone than novel or particularly complex knowledge-based work. These were factors also highlighted in the HTA, where engineers commented on the challenging and prestigious nature of complex, novel work.

However, the error analysis showed that, although complex and novel work was indeed more error prone than more straightforward work when considering the number of faults generated per unit of time or per lines of code, faults in the less demanding work actually seemed to represent a greater threat to the dependability of the SSI system. For instance, the error analysis found that only faults in straightforward design tasks actually survived three cycles of writing, checking and testing. In the work sample test the only common-mode errors, made identically by at least half the participants, were in supposedly 'simpler' code.

The engineers' comments, then, seem to be an example of the base-rate fallacy (e.g. Evans, Handley, Over & Perham, 2002). When performing any particular aspect of the SSI DPP there is a greater likelihood that an error will occur in knowledge-based performance (e.g. complex or novel work) compared to skill- or rule-based

performance. However, the vast majority of work is skill- and rule-based, so that when confronted with any particular fault in the SSI programme, that fault is much more likely to be the product of skill- or rule-based performance than of knowledge-based performance. The correct way to include information about the prior probabilities of events when making conditional probabilistic inferences was laid out in 1764 by Thomas Bayes. However, there is a great body of research on normative reasoning that has long recognised that people do not seem to make inference in such a 'rational' manner (e.g. Kahneman, 1973). Although first viewed as an example of irrational bias in human judgement, other authors have argued that these effects are partly methodological in nature. For instance, Gigerenzer (e.g. Gigerenzer & Hoffrage, 1995) has argued that it is the format in which the data on which the estimation task is to be performed which is of importance; information in the form of probabilities (e.g. a likelihood of .05) leads to neglect of the base-rate information, whereas the same information in the form of frequencies (e.g. a likelihood of 1 out of 20) does not lead so frequently to fallacious inference.

Alas the exploratory phase of this research project was not designed with this issue explicitly in mind, and therefore was not intended to be able to distinguish between the competing hypotheses as to exactly why the base-rate fallacy occurs. However, there are several points that do have relevance to this study.

The engineers that participated in the exploratory phase commented explicitly about the high probability of error when dealing with novel and complex SSI work. Based on empirical evidence from the *in situ* data collection in this study, these comments seem to be an accurate reflection of the likelihood of committing error when performing such tasks.

The same engineers commented that because of this high probability of error in novel or complex work relative to simpler tasks, these aspects of the task should be, and indeed are, allocated more resources (e.g. time, expertise) than more routine work. These jobs are also seen as more interesting and prestigious.

On the one hand, this unequal allocation of expertise is a necessary solution to a practical problem, at least with regard to particularly difficult work. That is, there is a minimum level of knowledge and expertise that is necessary to complete each task, and if this is not met then the task cannot be completed. A clear example of this was observed in the work-sample test, when the two least-experienced engineers could not make a reasonable attempt at completing the most complex parts of the task.

However, as long as the minimum expertise requirement is met, it is not clear as to whether the disproportionate allocation of other resources (e.g. time, number of staff) to complex work is of maximum benefit to the dependability of the system. This would depend in part on the relative amount of complex to standard code in each SSI, and the relative error-proneness of the work.

In the case of the work sample test, an average of four faults was generated per participant in the four lines of complex code that had to be written, giving an average of about one fault per line of code. In the standard code an average of seven faults were made in 120 lines per code, giving a probability of .06 of committing an error per line of code. Therefore the complex work seemed to be at least 16 times more error-prone than the standard work (1/.06 = 16.67) per line of code. However, there was 30 times the number of lines of code in the straightforward aspects of the task compared to the complex parts (120 lines / 4 lines = 30). So, although the complex parts of the task are much more error-prone than the more straightforward parts, they are also very much rarer, proportionately speaking, and so are likely to contribute fewer faults overall to the final SSI code.

The work sample was of course a contrived task, and the actual proportions of complex vs. standard tasks, as well as the relative error-proneness of each, will be much more variable in real work. However, it was devised by an experienced signalling engineer to represent a realistic piece of SSI DPP work, and highlights the issue of the actual vs. perceived importance of tasks in terms of the contribution to system safety. This was the case even though on average the work sample participants devoted much more time to the complex work compared to the standard work; the standard code

received on average 80 seconds per line of code whereas the complex code received 280 seconds, three and a half times more.

The situation is reversed if the task of the checker is viewed in the same way. Taking the figures above, we can be fairly sure that virtually every line of the complex SSI data will contain a fault. However, there is a reasonably low probability that any particular line of the standard code will contain a fault. From this point of view checking the complex code represents an easier task than checking the standard code; checking run-of-the-mill code could be likened to looking for a needle in a haystack, whereas checking complex code would be akin to looking for a nail in a matchbox.

The problem of how best to allocate resources to the SSI DPP, highlighted above, is compounded when also considering common-mode error. In the work sample test, more than half of the participants wrote identical faulty SSI code in four separate parts of the straightforward task, i.e. four common-mode errors. For one particular fault, *all* of the participants wrote *exactly* the same, incorrect, code. When shown these faults in the post-task debrief all of the participants recognised what they had done incorrectly, implying that this was skill- or rule-based performance that had gone wrong due to strong "habit intrusion" (Reason, 1990). The fact that so many of the participants made these errors strongly suggests that the problem lay not with the particular individuals involved but more likely with some aspect of the task. Therefore, even if more resources were allocated to the straightforward aspects of the writing task it is likely that some errors would continue to be made with high frequency. Because of the similarity between writing and checking, though, and the reduced amount of time that such straightforward work receives, it is also likely that the checking process would not be good at detecting these errors.

So, based upon the allocation of resources to work by perceived difficulty, it appears that the SSI DPP may not be organised to optimise the reduction of error commission or the promotion of fault detection. Additionally, even if resources were allocated more optimally (i.e. based on difficultly and *amount* of work), common mode error would mean that the value of any extra resources allocated would be diminished if the

extra resources were merely redundant repetitions of the same task (e.g. more time on task, or an extra repetition of a task, even if by a different person).

It was hypothesised at the end of the *in situ* exploratory phase that it was the diversity between the checking and testing tasks that protected the system from the shortcomings discussed above. Two fault detection methods that make qualitatively different demands upon human cognition, and that require different skills and knowledge, may be more likely to detect a greater range of faults than more similar methods because of reduced susceptibility to common mode error.

4.3 Need for Laboratory studies

The data from the task- and error-analyses were collected in a naturalistic work environment. While being externally valid, the lack of control and internal validity meant a number of factors may have biased and confounded the results. Most importantly, in the fault-log audit, if a fault had been detected by a checker it would then have been corrected and so be unavailable for subsequent detection by the tester. This meant that the number and type of faults within the SSI data would not be the same for checkers and testers.

It was found in the task analysis that expertise of the signalling engineer tended to increase from writer to checker to tester. This would also tend to confound any effect on fault detection performance due to task type.

In addition to the problem of internal validity, the ability to manipulate task factors was limited in the actual work environment. The safety-critical nature of SSI design meant that changes to equipment, procedures etc., could not easily be performed when 'real work' was being conducted. Commercial considerations militated against the collaborating signalling firms setting aside significant equipment or personnel resources to investigate manipulations of task factors using 'synthetic' work.

So, to complement the fieldwork studies a programme of laboratory experimentation was planned.

4.4 Aims of the laboratory programme

The general aims of the experimental programme were threefold: A) to confirm and validate the results of the field work; B) to further investigate the areas of interest that emerged from the task- and error-analyses, and; C) to develop the findings for application to non-SSI domains.

Regarding validation of the fieldwork, the biggest area of uncertainty to remain after the previous, predominately qualitative studies was that comprising the characteristics and relative efficacy of the checking and testing tasks. The fieldwork had provided useful preliminary results, but too many factors remained unknown to provide firm conclusions. The experimental programme would analyse the specific patterns of error detected by checking and testing, in terms of the quantity and characteristics of faults detected at each stage.

As regards extension of the fieldwork, the main theme that emerged was the potential of task and cognitive diversity to improve the resistance of the verification and validation stages of the DPP to common-mode error. Various aspects of diversity would be investigated: Task factors, e.g. the differences between task environment and fault types; and individual differences, e.g. abilities and mental models.

First, however, one of the more persistent individual fault types found resistant to checking, related to the use of sub-routes, will be investigated.

5 Sub-route labelling

It was noted in the task- and especially the error-analysis that a particular fault type was detected particularly poorly by the checking stage of the DPP. This involved barring two conflicting, or opposing, train routes from being set at the same time. An important element involved in this task is also used in many of the other functions found to be widespread in the fault logs, such as identity and labelling faults, and was commented-on by the task experts in interviews. The particular task element is known as "sub-route labelling". This chapter will explore the reasons why performing sub-route labelling is problematic.

5.1 Sub-routes explained

Any particular portion of the railway under the control of an SSI system is divided into sections of track. For any particular track section the SSI keeps a record of whether it is occupied by a train, and this information is used to inform the permitted movements of trains over the interlocking. However, as well as knowing whether a particular track section is occupied by a train or not, the SSI system also needs to make sure that the points that control train movements are in the correct position ahead of a train, otherwise the train may not follow its intended route or may be de-railed.

The way that this information is encoded in SSI is in the form of sub-routes. Sub-routes correspond to track sections, but as well as defining the identity of the section of railway to which they pertain sub-routes also specify the expected direction of a train over the track section.

Configurations of track sections are variable, so a rule is used to specify the mapping between the route of the train and a text label used to encode this information in an SSI-interpretable way (Figure 5).

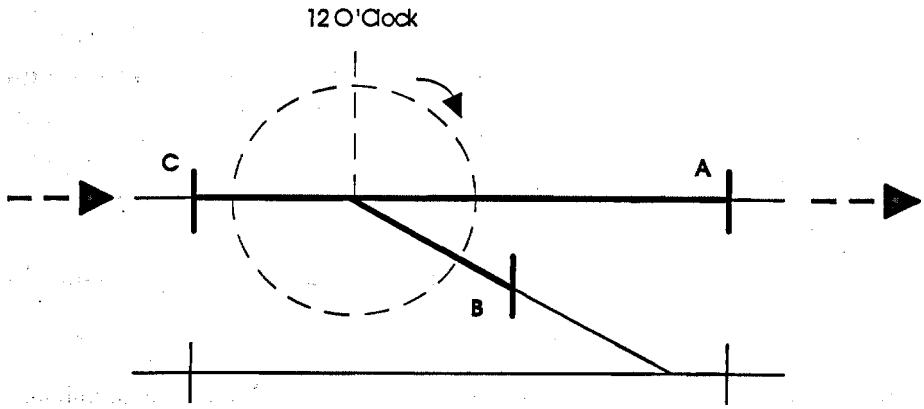


Figure 5: 12 o'clock rule for track section labelling.

Ends of track sections are denoted by letters (A, B and C), which is realised by imagining a clock-face centred on the conjunction of track ends, and alphabetically labelling the ends while moving clockwise from a 12 o'clock starting point. The label "CA" thus denotes a train movement in the direction indicated by the arrows.

Figure 5 demonstrates the currently used rule, called the "12 o'clock" rule. It shows a small portion (two track sections) of a highly simplified signalling plan. The parallel horizontal lines represent two adjacent main-line railway tracks, with a diagonal track connecting between them. There are sets of points at the intersections of the main and connecting lines, to guide trains along either the straight-line route, or across the connecting section to the opposite line. The railway lines are divided into sections, denoted here by the short vertical lines. There are thus two track sections shown, each with a set of points and three 'ends'. To demonstrate how the upper of the two sections (in bold) would be labelled, it has a clock-face superimposed over it, centred on the points. Moving clockwise from the 12 o'clock meridian, each successive end of the track section has been labelled in alphabetical order. So, for example, to denote the route between the dotted arrows, the label "CA" would be used. If this label were incorrectly specified (e.g. AC) a collision or derailment could potentially result.

As discussed briefly in the preceding chapter, this labelling procedure was found to be error-prone. Engineers committed errors when generating the labels, and more importantly when performing an independent check of others' labelling work.

Errors associated with sub-routes tended to be errors of commission rather than omission, i.e. a sub-route label would be present but incorrectly specified, rather than missing. This suggested that the problem with sub-routes lay not with the decision of when to use the sub-route labelling rule, but how it was applied or misapplied.

Two task-related factors were hypothesised that might influence observed performance. First was the mismatch between the alphabetical order of the label and the spatial direction of its corresponding route. With the twelve o'clock rule, all right-to-left routes have labels with ascending alphabetical order, and vice versa. For readers of the Latin alphabet, ascending order has a very strong population stereotype (Smith, 1981) of "left-to-right".

This population stereotype of alphabetical sequence has been found to affect rule-based task performance. Eikeseth and Baer (1997) used a matching-to-sample paradigm involving undergraduates learning relations between letters (e.g. A goes to B, F goes to M), their symmetrical opposites (e.g. B goes to A, M goes to F), and their transitive links (e.g. A goes to F, M goes to A). Errors were found to be increased when complexity was at its highest (i.e. deciding on a coexistent symmetrical and transitive relation), and also when the stimuli were adjacent letters in the alphabet; "next-letter bias" as it was termed.

These effects may be the result of a spatially-based mechanism underling the representation of alphabetical letter sequence. Gevers, Reynvoet and Fias (2003) found an association between letters earlier in the alphabet (e.g. A, D) and quicker responding to the left visual field, and letters later in the alphabet (e.g. X, Z) and responding to the right visual field.

This lack of compatibility between internal models and task demands may lead to response conflict and what Reason (1990) terms "strong-but-wrong" errors. A 6 o'clock rule, otherwise identical to the 12 o'clock version, would reverse the label alphabetical order and leave it compatible with the spatial direction, so removing this factor.

The second factor hypothesised to affect labelling accuracy was the spatial variability of applying the 12 o'clock rule. The first step of applying the rule is to site the imaginary 12 o'clock meridian over the points (if any) in the track section. The points are in different places in different sections, however, as can be seen from the upper (in bold) and lower track sections in Figure 5. Therefore before the labelling process can begin there must be a visual search of the (densely cluttered) track diagram to locate the appropriate point.

Teitlebaum and Granada (1983) found that inconsistently-placed menu elements on a computer screen increased visual search time by as much as 73% when performing menu-search tasks. It is hypothesised that inconsistent track section layout could be having a similar effect on sub-route labelling performance.

The track section / sub-route label positional inconsistency is also likely to increase working-memory demands during task performance. This is because the starting point for the labelling task has to be stored while labelling is proceeding and recalled if the task is interrupted even momentarily. Extra memory or processing demands have been found to increase reading times and error rates when reading aviation and industrial analogue displays (e.g. Grether, 1949).

Having the meridian at 9 o'clock would mean a fixed starting point for labelling in all cases, as there is always a horizontal component to the track section. This should represent a more consistent spatial mapping for applying the labelling rule, reducing the perceptual/attentional demands of the sub-route labelling task.

Two experiments simulating label-checking were thus undertaken to test these hypotheses, comparing label checking performance between: 1) 12- and 6 o'clock rules; 2) 12- and 9 o'clock rules. It was expected that sub-route label-checking performance would be improved (quicker, more accurate) when using the revised rules than when using the traditional 12 o'clock rule.

In addition to measures of task speed and accuracy it was thought necessary to measure how hard the participants had to work in order to achieve their level of

performance. Various authors (e.g. Hockey, 1997) have shown how a given level of performance on a task can be maintained at the expense of a greater investment of effort, even though sustained effort will lead to fatigue and task decrement over time.

5.2 Sub-route experiment I: 12 o'clock rule vs. 6 o'clock rule

5.2.1 Method

5.2.1.1 Participants

Novice participants were chosen for this and the subsequent experiment because experienced signalling engineers would already by highly practiced at using the 12 o'clock rule, so biasing the relative performance between the old and new labelling rules.

Thirty-three novice participants (23 female, mean age 24 yrs), an availability sample recruited on a UK University campus, each performed two within-participants labelling conditions (12 o'clock vs. 6 o'clock rules).

5.2.1.2 Materials

A set of 96 "test" track section diagrams were produced for display on a VGA computer screen. Each diagram consisted of:

- One of the four possible different track section "shapes" of the type shown in bold in Figure 5 (made by either a horizontal and / or vertical reflection of the original Figure 5 shape)
- One of the four possible different train directions that could travel over the track section, denoted by arrows as in Figure 5
- Below the track section, one of the six possible different two-letter sub-route labels (e.g. AC, BA).

Additionally, a set of 32 "practice" diagrams were produced, using each of the four different track shapes combined with each of the four different train directions, but this time showing the correct sub-route label for half and an incorrect one for the rest.

Unlike Figure 5, none of the diagrams contained a circular "clock face" feature – the stimuli were designed to look like actual (but highly simplified) sections of track signalling layout maps.

Each diagram was approximately 60 mm high by 80 mm wide when displayed. Lines were approximately 2 points wide, and the label text presented in 20 point Times New Roman font. Blue was used as the background colour on the computer display, white as the diagram/ text colour.

A paper-based sub-route diagram very similar to that in Figure 5, along with a brief explanation, was produced to teach the 12 o'clock and 6 o'clock rules to the participants.

The NASA TLX multi-dimensional workload scale (Vidulich and Tsang, 1986) was used to provide an overall workload score associated with task performance.

The practice and test stimuli, and the NASA TLX workload scale, were programmed for display on an IBM PC compatible computer using Borland C++. The computer also recorded the participant's response key press and reaction time.

5.2.1.3 Procedure

Participants were required to complete the label checking task using both the 12 o'clock and 6 o'clock rules, and so they were first randomised to either an AB (12 o'clock rule first) or BA (6 o'clock rule first) counterbalance sequence.

Each condition then followed the same pattern. First, the relevant labelling rule was explained to the participant using the explanatory diagram. The participant was then presented with the 32 practice stimuli one at a time in random order on the computer. When each stimulus appeared on screen the participant had to decide if the train direction shown on the diagram correctly corresponded to the label shown below the diagram, by applying the current sub-route labelling rule. Responses were forced-choice: if the participant thought the label was correct they pressed the "Z" key, if they

thought it was wrong they pressed the "." key. No feedback regarding the correctness of responses was presented.

At the end of the practice stimuli the participant was required to go through the practice session again if they had failed to achieve a level of 75% accuracy on the practice items.

After the practice had been successfully completed the participants were given the opportunity to ask questions before moving on to the test stimuli. When ready, the participants were told to complete the coming task using the labelling rule they had been taught "as quickly as you can without making mistakes". The 96 test stimuli were then displayed one at a time in random order in an identical fashion to the practice stimuli.

When all test stimuli were finished the NASA TLX was displayed and completed by the participant. Finally, the procedure described above was repeated for the other labelling rule.

5.2.2 Results and discussion

One participant's data were removed due to incorrect completion of the NASA TLX. All statistical tests were two-tailed, $\alpha = .05$. Table 11 shows the time of correct responses, accuracy and workload mean scores by labelling rule used. The 6 o'clock labelling rule was somewhat more error-prone (by 1.3% on average) than the 12 o'clock rule, with the response time and workload scores showing very small differences between rules. A multivariate, repeated-measures analysis of variance was conducted, with labelling rule (12 o'clock vs. 6 o'clock) as the repeated independent variable and error rate (%), time for correct response (s) and NASA TLX workload score (0 = "minimum", 100 = "maximum" workload) as dependent variables.

Table 11: Performance measures by labelling rule (12 o'clock rule vs. 6 o'clock rule)

Performance measure	12 o'clock rule	6 o'clock rule
	M (*SD*)	*M* (*SD*)
Error rate (%)	4.00 (3.60)	5.30 (6.30)
Time for correct response (s)	4.19 (0.95)	4.35 (1.12)
TLX workload score (maximum score=100)	60.00 (15.00)	59.20 (14.90)

Note: N = 32

There was no significant difference found between conditions ($F [2, 30] < 1$). A repeat of the analysis with counterbalance order included as an additional between-participants independent variable also showed no significant effect.

The results suggest that, at least in this circumscribed context, the mismatch between label stereotype and spatial direction does not have a significant effect on speed or accuracy of performance, nor on how difficult the tasks are perceived.

5.3 Sub-route experiment II: 12 o'clock rule vs. 9 o'clock rule

5.3.1 Method

The method for this experiment was identical to that for experiment one, barring the substitution for the 9 o'clock labelling rule here for the 6 o'clock rule used in experiment one.

5.3.1.1 Participants

Sixteen novice participants (nine female, mean age 23 yrs) were recruited by opportunity sampling at the same UK University campus as used in experiment one. Each participant performed two within-participants labelling conditions (12 o'clock vs. 9 o'clock rule) in the same way as for experiment one.

5.3.2 Results

Table 12 shows the performance measures (mean scores of time of correct response, accuracy and workload) associated with sub-route labelling-

Table 12: Performance measures by labelling rule (12 o'clock rule vs. 9 o'clock rule)

Performance measure	12 o'clock rule	9 o'clock rule
	M **(*SD*)**	***M*** **(*SD*)**
Error rate (%)	7.30 (4.00)	8.30 (5.50)
Time for correct response (s)	4.21 (1.04)	3.32 (1.02)
TLX workload score (maximum score=100)	63.50 (11.00)	55.10 (11.40)

N = 16

The 9 o'clock rule labelling condition showed a slightly higher error rate (8.3% vs. 7.3%) and lower time for correct response (3.32 s vs. 4.21 s) than the 12 o'clock rule condition. Workload was also somewhat lower for the 9 o'clock rule labelling condition (55.1) compared to the 12 o'clock condition (63.5).

An identical analysis to that carried out for experiment one was performed on the data from experiment II: a repeated measures multivariate analysis of variance with rule

condition as the independent variable and speed of correct response, error rate and workload scores as the dependent variables.

In this case, however, there was a significant difference between conditions ($F [2, 14] = 6.84, p < 0.01$). Subsequent univariate analysis showed mean time for correct response ($F [1, 15] = 22.3, p < 0.01$) and NASA TLX score ($F [1, 15] = 11.3, p < 0.01$) both individually differed significantly across conditions, with the 9 o'clock rule producing better performance in each case. Again, the analysis was repeated with counterbalance order included as a between participants independent variable with virtually identical results.

To check for possible speed-accuracy trade-offs the times for correct responses were correlated with accuracy for both the 12- and 9 o'clock conditions. The obtained coefficients, $r (16) = .15$ and $r (16) = .14$ respectively, were not significant. "Micro" speed-accuracy trade-offs (e.g. Rabbitt, 1966) were checked by comparing the mean time for correct responses with the mean time for incorrect responses within condition. There were no significant differences for either the 12 o'clock ($t [15] = .48, p > .05$) or the 9 o'clock ($t [15] = .79, p > .05$) conditions.

5.4 General discussion

The hypothesis that the mismatch between route spatial direction and label alphabetical order leads to poor performance on the safety-critical route labelling task was not supported (experiment one). However, it seems that using a spatially inconsistent vs. a spatially consistent labelling rule does have an effect on performance (experiment two).

The lack of support for the population stereotype mismatch hypothesis may suggest that the labels generated by the various rules are not viewed as sections of an ordered alphabet, but rather as abstract, two-letter acronyms, where the concept of alphabetical-order has no reliable association. Even if viewed as part of the ordered alphabet, there is only the smallest possible 'distance' between the stimuli used in this experiment, i.e. A, B and C.

It could be argued that use of "6 o'clock" as opposed to "12 o'clock" represents a much weaker population stereotype relating to clocks, leading to less efficient use of the analogy, and a possible confounding factor with changes in performance due to alphabetical order. However, the 9 o'clock rule used in experiment two would not seem to represent a stereotypical starting point for reading a clock face any more than the 6 o'clock rule, but performance in that condition was significantly better than in the standard 12 o'clock rule condition.

In experiment two, participants were significantly faster in making correct responses for the 9 o'clock labelling condition. The lower workload ratings suggest that this is at least partly due to them finding this condition less difficult. The lack of a significant difference in error rates may be due to a lack of statistical power, probably confounded by the restriction of range in the scores as the participants concentrated on achieving accuracy. Although there was no significant difference between the error rates for the two conditions, the 9 o'clock rule was more error prone than the 12 o'clock rule. This did not seem to reflect an underlying speed-accuracy trade off, however, as speed and accuracy were not well correlated, and correct responses were not significantly slower than erroneous ones.

Although participant numbers for the second experiment were quite low there was a good similarity between participants' performance while using the 12 o'clock rule across the two experiments: average error rate was within 4% and average response time differed by less than .1 second.

There is some question over the generalisability of the results from these experiments. The participants in both were novice to the practice of sub-route labelling and therefore the reasons underlying variations in their performance may not hold true for experienced signalling engineers. This criticism is undeniable but it is mitigated by a number of factors.

First, an experiment using experienced SSI engineers would not have allowed a fair test of the new labelling rules as responses would have inevitably have been biased by extreme familiarity and experience in using the existing 12 o'clock rule. Second,

important characteristics of the participants in these experiments are likely to be broadly similar to the ones seen in the work sample test (§ 3.1). The signalling engineers were found to have higher levels of cognitive abilities (e.g. vocabulary, clerical perception) than the norm, and this is also likely to be true for samples of University students such as the participants in these experiments.

Based, as this task is, on the work of designers of safety critical systems, errors and their reduction would seem to be of utmost importance. However, the ability to work faster while reporting less overall workload clearly shows a lower demand for limited mental resources (e.g. Wickens, 1984) when using the 9 o'clock labelling rule. From a practical point of view, for instance, this means that the engineer will be less vulnerable to the distractions and interruptions ever present in the open work environment in which the SSI DPP is conducted. Perhaps more importantly, it means that the designer will be better able to manage the complex, "knowledge-based" (Rasmussen, 1980), higher-level task elements that make up the "bigger picture" of design tasks, if they are not getting bogged down with the details.

The greater efficiency of the 9 o'clock labelling rule demonstrates the importance of the way tasks are represented to the operators. Even the minor modifications used in these experiments demonstrated measurable benefits in relatively simple implementations of the actual safety-critical signalling tasks.

The goal of the following chapters will be to look at what effect diverse task representations can have on qualitative as well as quantitative aspects of performance, i.e. not just how many errors, but what type of errors are more likely with different representations, and how this can be used advantageously by the system.

6 Task and Cognitive Diversity

The concept of cognitive diversity is related to the practice of using multiple components in a system. The use of multiple components to improve the fault tolerance of systems is well established (see § 1.1.2). In its most common manifestation this involves the use of two or more identical system components to perform the same function. If the failure modes of the component are known, the system can be designed with one of the components off-line; if the master component should then fail, this can be detected and the off-line component can take over system functioning. If the failure modes of the component are not known, then the multiple components can run in parallel with their outputs compared with one another. If the different components do not produce the same output given the same input, at least one is assumed to have failed.

This use of multiple component redundancy can guard against failures due to essentially random processes, such as radiation altering the state of a bit in memory, or faults in manufacture or fitting. However, identical components will be equally susceptible to conceptual errors in their design or use. These 'common-mode' errors occur because of the lack of diversity in performance of each one; they share the same strengths but also the same weaknesses. To combat this lack of 'product diversity', the concept of 'process diversity' can be applied during the design of components. When applied to the human elements of the system, this can be conceptualised as 'cognitive diversity'.

Redundancy of human components has been demonstrated to have beneficial effects in programming tasks. For instance, Bisant and Lyle (1989) demonstrated improvements when two people engaged in code checking. Wilson, Nosek, Hoskin and Liou (1992) found improved problem-solving for teams of programmers; the principle demonstrated was that if one individual 'fails' in some aspect of a task, then a co-worker may be able to detect the mistake. However, as demonstrated in the work-sample task from the error analysis (§ 3.1.2.2), different individuals can be susceptible to common-mode error.

Although human components will not be identical in the same way that redundant mechanical components can be, they may share similar performance capacities, knowledge and strategies that result in common-mode error. For instance, uniform selection and training practices within an organisation will serve to reduce the variety of approaches to a task. As asserted by Senders and Moray (1991), "The use of multiple humans in the way multiple inaccurate components are used is not a reliable way to enhance human-machine system reliability".

One way to combat this problem is to increase the diversity between human components in the system. Fagan (1986) found that if software inspection team members are diverse in terms of task perspective, software quality can be improved. A more general approach to diversity can be taken from a model of human-dependent failure (HDF) put forward by Hollywell (1993). Here, HDFs arise when 'Root Causes' (e.g. distraction, working memory limitations), impact on human actions through 'Coupling Mechanisms' (e.g. task environment, training). In effect, the coupling mechanism "creates the conditions for multiple human actions to be affected by the same root cause". Defences against HDF can be directed at the root cause and/or the coupling mechanism. Addressing the root causes, through the application of Human Factors knowledge, will serve to increase the overall 'quality' of the system. However, complete eradication of error is not likely to be possible (e.g. Frese and Zapf, 1991). Another approach would involve increasing the diversity of coupling mechanisms so that human performance is not so narrowly and directly related to root causes. This approach underlies the use of task and cognitive diversity to reduce system vulnerability to common-mode error.

6.1 Task and Cognitive diversity applied

In plain terms, the use of diversity does not seek to reduce the number of errors committed by personnel. Rather, it seeks to ensure that, by the use of varied coupling mechanisms, the root causes of error are manifest in different ways, or types of error. The reciprocal of this error-focused description is that correct performance will differ in its characteristics across different coupling mechanisms.

The following sections of the thesis will be concerned with fault finding performance between the checking and testing stages of the SSI DPP. Checking and testing represent diverse coupling between human capabilities and task performance by the way the tasks are represented. Task diversity and the related concept of 'cognitive diversity' that it engenders are discussed by Westerman, Shryane, Crawshaw, Hockey and Wyatt-Millington (1995) and Westerman, Shryane, Crawshaw and Hockey (1997).

Figure 6 represents diversity by use of a Venn diagram. The outer rectangle represents the set of all faults in an SSI geographic data programme (A). The shaded ellipse (C) represents a subset of A containing the set of faults that could possibly be detected during checking ($C \subset A$). The dotted ellipse (T), also a subset of A, contains the set of faults detectable during testing ($T \subset A$).

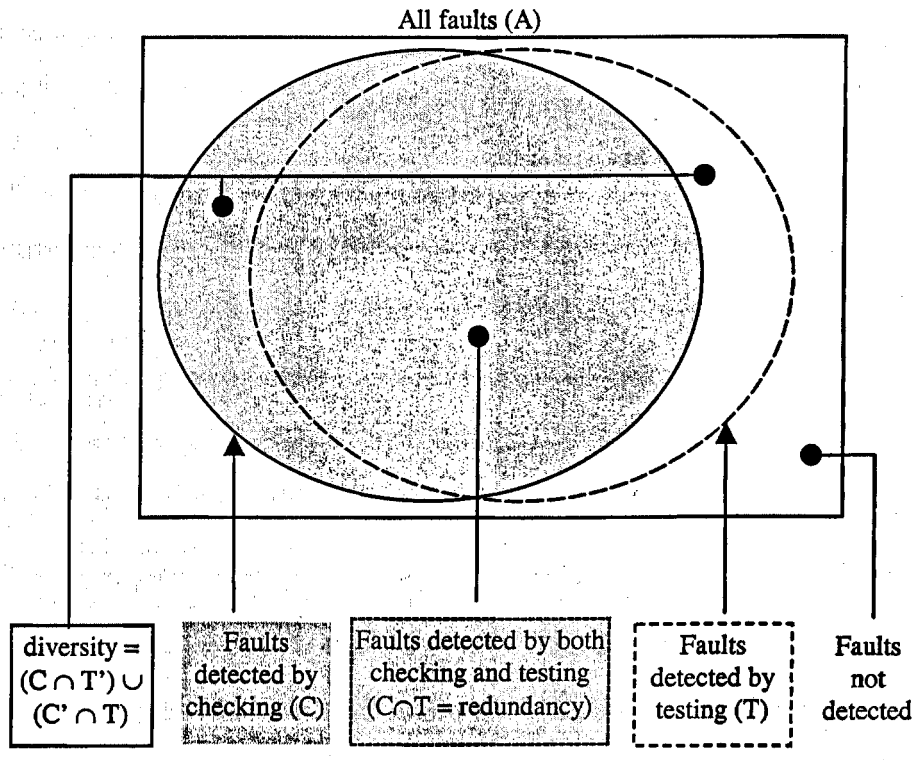


Figure 6: Schematic of diversity between checking and testing tasks in the SSI DPP.

The area in the middle, where C and T overlap, contains the set of faults detectable during both checking and testing ($C \cap T$). This middle portion represents redundancy

The area in the middle, where C and T overlap, contains the set of faults detectable during both checking and testing ($C \cap T$). This middle portion represents redundancy in the system, where, if a check failed to detect the fault, it would still be potentially detectable in testing. The area of C that is not also in $C \cap T$ represents those faults detectable only during checking, and so represents diversity in the system. Similarly, the area of T not also in $C \cap T$ represents those errors detectable only during testing, another instance of diversity in detection. In total, diversity can be represented as $(C \cap T') \cup (C' \cap T)$.

For simplicity, in the example above faults were considered to be detectable or not by either task. This is gives conveniently defined boundaries to the various ellipses, because a procedure is considered to be able to detect a fault in principle, or not. However, in the real world of the SSI DPP very few types of faults are entirely undetectable in one of the two tasks. For most real faults there will be some non-zero probability that it will be detected by a 'typical' engineer working on a 'typical' job. Faults are then characterised as being more or less 'easy' or 'hard' ('visible'?), depending on whether the probability associated with their detection is high or low. However, for the purposes of the explanatory model, the case as described conveys the main thrust.

There are at least two factors that must hold to be true before the use of diversity, as shown in Figure 6, can be of benefit.

First, there must be some constancy to error detection; there should be a somewhat consistent relationship between error detection and task type. More formally, we should expect that the distribution of detected errors is not random across task type. That is, the circumstances that promote error must be related to the nature of that error. This assertion is well supported in the literature (e.g. Reason, 1990).

Second, neither of the fault detection methods should be a subset of the other. If it were possible for checking or testing to consistently locate all of the faults that were detected by the other method, again there would be no requirement for diversity (in Figure 6, $C \not\subset T$ and $T \not\subset C$) – use of the subordinate method would be futile. (There is

a special instance of this requirement, where one detection method is perfect and always detects all of the faults. In this case there would be no need for a second method as it would just be a subset of the perfect method).

Finally, it is worth pointing out that diversity as conceptualised here does not depend on quantitative differences between fault detection strategies, but rather on qualitative differences; that is, it's not the *number* of errors detected by each method that is of prime importance, but whether or not they are of a variety of fault detected well or poorly by the other method.

For example, in Figure 6 the ellipses representing checking and testing are drawn of equal size, suggested that C and T are equally effective at fault detection when applied singly. However, even if C was half the area of T, that is, able to detect only half the number of faults, it could still be highly worthwhile applying both C and T as long as C detected faults that T missed.

6.2 Dimensions of diversity

Figure 6 illustrates the use of diversity with reference to variety in task environment. This is only one of the many dimensions that could be exploited in the drive to reduce common-mode error.

6.2.1 Task environment

In the case of the SSI DPP, checking and testing represent diverse task environments. As discussed previously, checking can be characterised as a textually-based, abstract task. Testing can be characterised as a spatial, concrete task. These and other differences are likely to lead to differences in performance characteristics, as different methods and modalities of information presentation have been shown to lead to different perceptual and cognitive biases (e.g. Baddeley, 1986; see Wickens & Hollands, 1999, for a review). The saliency of information presented to the operator may also be very different in various task environments (Rasmussen and Lind, 1981). This will impact on the types of errors committed directly, but also indirectly by the encouragement and support of different mental models and performance strategies.

6.2.2 *Domain Knowledge*

Domain knowledge is likely to be an important factor in cognitive diversity, but one that will have benefits and drawbacks depending on the extent of the difference. If two methods / individuals share identical domain knowledge, then they will be highly susceptible to common-mode error. On the other hand, if the two share no domain knowledge then this can be seen as specialisation, with no overlap in their combined performance. Domain knowledge will be closely related to, but not necessarily defined by, task environment.

6.2.3 *Performance strategy*

The nature of the checking task means that engineers deal with the 'nuts and bolts' of the SSI data language, from which they construct the railway signalling functionality. This is likely to encourage 'bottom-up' performance strategies. Testing is the reverse, as engineers deal directly with the high level functioning (e.g. setting a route), and then deconstruct this into its constituent parts (e.g. point movement, signal colour change). This is likely to encourage 'top-down' performance strategies. Research has shown that there are quantitative differences in fault-finding performance associated with different performance strategies. Morrison and Duncan (1988) found that 'top-down' strategies are more cognitively demanding than 'bottom-up' ones, but potentially more effective. In the context of diversity, it may be that the various strategies differ in the characteristics of their performance (e.g. fault types detected) as well as their effectiveness (e.g. number of faults detected).

6.2.4 *Individual characteristics*

There are a number of dimensions of individual difference that might be useful in terms of cognitive diversity, e.g. personality, cognitive ability, cognitive style. Cognitive ability in particular has been found to be associated with programming skill (e.g. Egan, 1988), and in the work sample test (§ 3.1) spatial ability was associated with better performance. Studies addressing this issue have been concerned with predicting absolute levels of performance. However, when considering diverse systems of verification and validation what is also of interest is the association between diversity of ability and diversity in the characteristics of fault-finding performance.

For instance, it has been proposed that a distinction exists between the mental representations adopted by individuals while performing problem solving tasks (see MacLeod, Hunt & Matthews, 1978). Participants high in spatial ability will tend to use mental representations that are essentially spatial, while others, high in verbal ability, will tend to use verbally-based representations. If different faults are more differentially conspicuous with different representations, diversity in fault finding should be apparent.

Another area of promise for diversity is that of cognitive style. While measures of cognitive ability are normally concerned with maximum performance, cognitive style "...implies the measurement of propensities in terms of typical performance with the emphasis on a predominant or customary processing mode" (Tiedeman, 1989, p.263). Previous research in fault finding performance (Morris and Rouse, 1985; Morrison and Duncan, 1988) has found benefits associated with a reflective rather than impulsive style, and with analytic as opposed to global styles.

The concept of cognitive style has been criticised by some researchers (e.g. McKenna, 1984), who suggest that styles are strongly related to aspects of cognitive ability, and so not really universally available 'styles' of processing at all. However, in the context of diversity, any dimension that leads to predictable variation in an individual's performance characteristics across tasks will be of interest. As long as the two factors underlying the benefits of diversity are met (§ 6.1), differences in terms of absolute performance are not as important as the characteristics of that performance.

6.2.5 Mental models

A mental model has been defined as "...a rich and elaborated structure, reflecting the user's understanding of what the system contains, how it works, and why it works that way" (Carroll and Olson, 1988, p. 51). An individual's mental model of a system will depend somewhat on the extent and accuracy of their domain knowledge, and also the task representation. However, the concept of a mental model also includes the structure and inter-relationships of the information it contains, and so even if two individuals could be found with identical 'knowledge' of a system there could still exist

extensive variation between their mental models. For these reasons differences in mental model could be a valuable source of cognitive diversity.

A programme of three experiments investigating cognitive diversity was therefore enacted. These experiments, described next, will be used to study the use of cognitive diversity as a method for improving fault detection performance.

7 Cognitive Diversity Experiment I

The first experiment investigating cognitive diversity in the SSI DPP was designed to confirm (or otherwise) the findings of the previous task- and error-analyses, and to investigate factors of diversity associated with fault detection performance as outlined in the previous chapter.

As was done for the fault log audit, the differences in between the checking and testing stages of the DPP would be examined. Specifically, it was hypothesised that the two stages would differ in their effectiveness in detecting different types of fault, thus demonstrating diversity through task environment. For this experiment, however, some of the major confounding variables present in the work-based data collection would be controlled.

It was also hypothesised that differences in people's relative spatial and verbal abilities may lead them to adopt a predominantly spatial or verbal style of processing. It was hypothesised that a spatial processing style would be most advantageous in the testing task, and a verbal style most advantageous in the checking task. This would lead to greater fault detection performance if the individual's style of processing matched their task environment. Differences in style would also be expected to lead to diversity in the type of faults detected by each processing style, which would be manifest as an interaction between style and task environment.

7.1 Method

7.1.1 Participants

It became apparent after the error analysis work-sample task that it would not be possible to obtain sufficient numbers of trained SSI engineers to participate in experiments. Fifteen engineers took part in the work-sample test, and while this was low for statistical purposes, it represented a significant proportion of all of the suitably qualified candidates in the UK. While this was sufficient for exploratory work, numbers per cell would drop too low if the sample was divided into treatment groups, as would be the case with the laboratory studies.

As for the sub-route experiment (see § 5) naïve participants would be used. This would not have the external validity of using SSI engineers, but naïve participants would not be biased by conventions in task performance (e.g. methods, strategies, expectations) as the experienced engineers would, and their level of experience would not be confounded with the type of task that they were most experienced in (testers tend to be more experienced than checkers in the SSI DPP). The naïve participants would all be matched in terms of SSI experience, i.e. *ab initio*, but to allow for this lack of expertise, simplified versions of real SSI tasks were developed.

Therefore, 27 participants with an engineering or computer science background were recruited from the student population at a UK University. Engineering and Computer Science students were chosen as a more homogeneous sub-set of participants in terms of their experience of de-bugging software compared to the general student population. Additionally, less training would need to be given in the basic features of fault-detection tasks, so more time could be spent on SSI-specific training.

The participants were randomly allocated to one of two conditions. In one condition the participants were to perform the task of SSI checking (n = 13; mean age = 24 yrs). In the other condition the participants would perform the task of SSI testing (n = 14; mean age = 23 yrs). All of the participants were male.

7.1.2 SSI simulation

To enable the aims of the experimental programme to be fulfilled, simulations of the DPP checking and testing tasks were developed. Because the participants for the experiments would have no SSI experience the tasks chosen for simulation were only a sub-set of the full range of work that would normally be carried out in designing an SSI. Specifically, only the functioning associated with the setting of routes across the railway network would be simulated. Faults would then be seeded into the simulations, and the detection performance of checkers and testers could be compared.

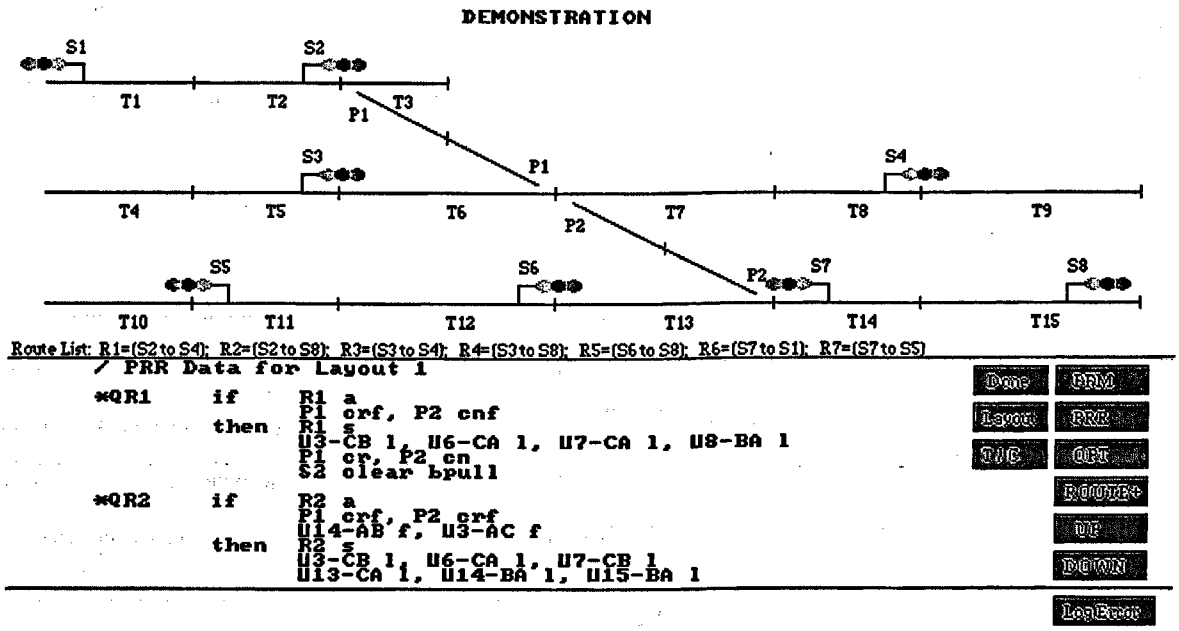
Although checking is predominantly paper-based and testing computer-based, it was decided to build a common computer interface for simulations of both tasks. This would allow the differences in interaction methods for the two tasks to be controlled

and assure that task-related information presentation was consistent across checking and testing tasks.

The SSI simulator was programmed in Borland C++, and runs on IBM-compatible PCs under MS-DOS. Control of the simulator is via mouse for screen elements and keyboard to type in details of faults found.

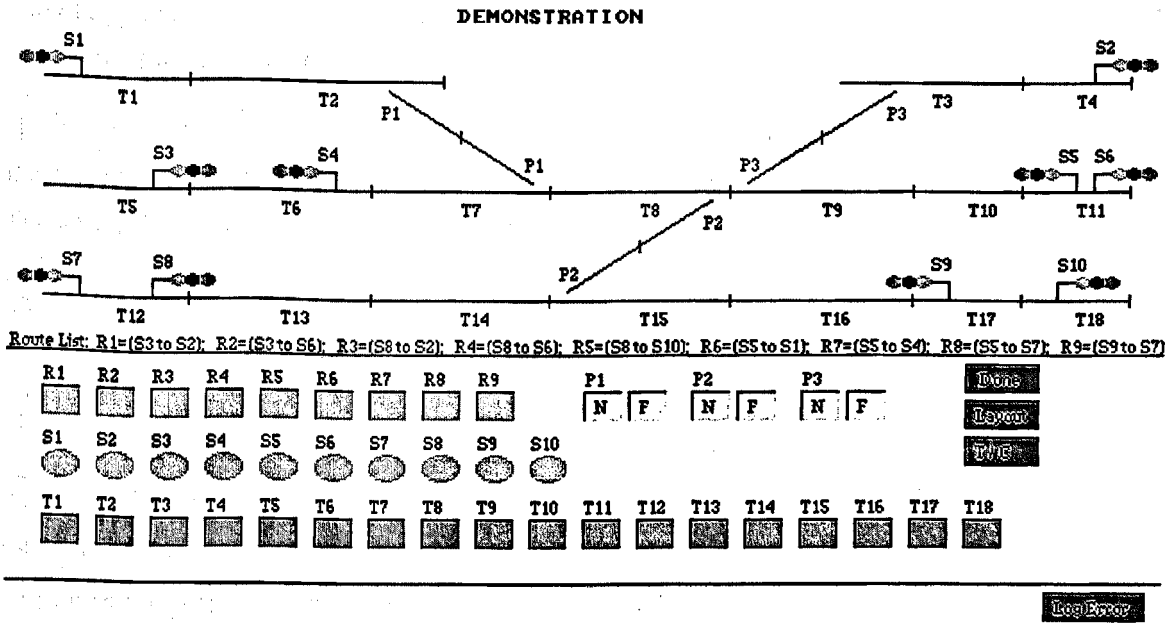
The simulator computer-screen was divided in half horizontally. For both tasks, signalling layout diagrams were shown in the upper half of the screen. These included the tracks, points and signals and also a list of the possible routes in the layout. For the checkers, the lower half of the screen showed print-out of the SSI Geographic Data files; the testers instead had an array of controls (e.g. for points) and indications (e.g. of signals) to enable the testing of the behaviour of the system. There were three track layouts programmed into the simulator, one for training and two for actual performance: the simple layout used in training contained only four routes; the two layouts for data collection contained seven and nine routes, respectively. The screens for checking and testing tasks, showing the seven- and nine-route layouts, are shown in Figure 7 and Figure 8, respectively.

Figure 7: SSI simulator checking screen



Note: The upper part of the screen shows the signalling plan for the simulated SSI scheme, including a list of routes. The scheme shown here is the simpler of the two used in experiments I, II and III, with seven possible routes (only four of which were to be checked). The lower part of the screen displays the SSI 'geographic data' code to be checked. Navigation between and within the different SSI 'files' is achieved by use of the buttons on the right hand side.

Figure 8: SSI simulator testing screen



Note: The upper part of the screen shows the signalling plan for the simulated SSI scheme, including a list of routes. The scheme shown here is the more complex of the two used in experiments I, II and III, with nine possible routes (only four of which were to be checked). The lower part of the screen displays the controls and indicators that are used to test the functionality of the SSI 'geographic data' code. Route controls are labelled 'R', track sections are labelled 'T', points 'P' and signals 'S'.

7.1.2.1 Checking task

The simulator requires the participants to search for faults in SSI programme code concerned with setting routes across the railway network. Checkers do this by browsing through three 'files': PRR, PFM and OPT, each accessed by clicking on the respective button. The 'PRR' file (Panel Route Request) contains the SSI data specifying the conditions that must be fulfilled before a route can be set, e.g. points in their correct positions, no opposing routes already set. The PFM (Points Free to Move) specifies when it is safe to move the points from one position to the other (e.g. no train travelling over them). The OPT (OutPuT) file ensures that only one train is allowed into a route at a time, by controlling when a signal is sent (output) to change the route's entrance signal to green.

The checkers search for faults by reading through the SSI code, ensuring that it complies with the signalling rules that apply to the particular SSI signalling layout shown in the diagram in the upper part of the screen.

7.1.2.2 Testing task

Testers search for faults by making sure that safe actions can be carried out but unsafe ones are not allowed. The lower part of the simulator screen contains buttons corresponding to each of the screen elements, e.g. points, signals and track sections. Each one of these can be toggled to different states, e.g. points can be in one of two positions, and can be free to move or locked; track sections can be empty or have trains situated in them. For example, a prime safety concern is that points should not move from one position to the other while a train is crossing over them as the train would be de-railed. This is tested by taking a section of track containing a set of points, and then setting the state of the track section to represent a train situated within the track section. The points should be fixed in position, unable to be moved until the track section has its state changed to represent no train in the section.

7.1.2.3 Faults to be detected

The faults to be detected by the participants were seeded into the simulators. To ensure that performance by checkers and testers was comparable exactly the same set of faults was used for both simulators. In the real SSI checking and testing tasks, some fault types exist that would not be detectable by both methods. For instance, duplicated code may be visible in checking, but would not necessarily affect specific functionality of the railway during testing, and so be invisible. Only when the data were installed on-site could the extra processing demands of the superfluous code lead to system failure. Other problems, such as timing constraints, may be input-dependent and so not detectable by checking the SSI code alone. To avoid these problems only faults that would in practice be detectable by both checking and testing were included.

Faults from four signalling categories were chosen to be seeded into the simulations, based upon common fault types found in the error analysis:

1) ASPect control (ASP). Faults in this category affect whether a green light, or aspect, is shown appropriately by a signal

2) Opposing Route, Same points position (ORS). The SSI system should not allow a route to be set if another route which uses similar parts of the track network (an opposing route) has already been set. Some opposing routes require the points to be in the same position for both routes; other opposing routes require points in different positions. The SSI system deals with these two classes of opposing routes in different parts of the SSI code. This category deals with faults affecting opposing routes over the same points' position.

3) Opposing Route, Different points position (ORD). This category deals with the second class of opposing routes, those set over different points positions. Both of these opposing route categories were used because it was thought that the different methods of coding may lead to performance differences between categories for checkers.

4) ROUte setting (ROU). This category deals with faults affecting whether a route can be set appropriately, e.g. ensuring that the sets of points required by the route are moved to the correct position.

These fault categories were chosen from those recorded in the fault log audit section of the error analysis chapter (§ 3.2), because of either their high frequency of occurrence or because of large differences between the performance of checkers and testers. These fault categories thus represent an attempt at a robust manipulation of the fault-type variable, so increasing the statistical power of an otherwise "small" experiment.

7.1.3 Procedure

Data collection took place in groups of up to 10 participants at a time. Each participant worked individually, seated at a PC workstation running the SSI simulator.

All participants initially completed tests of verbal and spatial ability taken from the General Aptitude Test Battery (US Dept. of Labor Employment and Training Administration, 1982). This was followed by two periods of training lasting approximately 45 minutes each.

In order to standardise the training for the tasks between checkers and testers as far as possible all participants first completed a 'core' component which related to the general working of SSI and railway signalling rules. This was followed by specific training in which each experimental group was taught how to use the particular interface associated with their task to find faults in the simulated SSI. Between training sessions, and between training and testing, participants were given 5-minute refreshment breaks.

During training the SSI simulator was running a simplified, practice SSI track layout. For the core training participants were presented with an information sheet, and this was read through by an experimenter who explained the various concepts to the participants and answered any questions that they had. The participants also had a crib sheet to use while performing the training and actual task. The training and crib sheets can be found in appendix B.

The specific training included a practice session, where participants searched for two faults in a demonstration layout of the experimental task presented on the SSI simulator. At the end of the specific training session the practice faults were shown to the participants, along with the reasoning and actions that should have been followed to find them. Questions about the training, e.g. the signalling rules, were fielded from the participants at this point; the participants were told that questions could not be answered once the task proper had begun.

For the actual task participants were required to complete two signalling layouts, presented one after the other. One layout consisted of seven train routes; the other, nine. Presentation order of the two layouts was counterbalanced within conditions. For each layout participants were instructed to check or test the code / behaviour of four of the routes (specified on the crib sheet) and to search for any faults that might be present. Just four routes were specified for the task because as the number of active routes increases linearly the interactions between routes increase exponentially. Thoroughly testing more than four routes would have made the data collection sessions too long. Although only actually looking for faults in four of the routes, the extra routes present in each layout would represent additional information and clutter, thus making the task more demanding and avoiding ceiling effects.

Each layout contained eight faults, but the participants were not informed as to how many faults there were to find. There were four fault types (ASP, ORD, ORS and ROU, as described earlier), two of each type per layout. Further, for each fault type, one of the faults per layout was an error of commission, i.e. incorrect SSI code, and one was an error of omission, i.e. missing code.

If the participant found a fault they were instructed to log it by clicking a button on the display and inputting a description allowing the fault to be identified. The task was self-paced. Participants were instructed to continue searching for faults until they were satisfied that they could find no more. When they had finished inspecting a layout they clicked on another button on the display, which either took them to the next layout, or terminated the simulator programme. Participants were paid £15 for completing the experiment.

7.2 Results

The data from one of the participants in the checking condition was excluded from the analysis because of misinterpretation of experimental instructions.

There was no significant difference between checking and testing groups in spatial ability. However, the testing group scored significantly higher ($t\ [24] = 2.24$, $p < .05$) in verbal ability. Correlations between the measures of cognitive ability and fault detection performance were examined separately for each group. No significant associations were found.

To test whether differences in spatial and verbal ability were related to error detection performance, a measure of relative spatial / verbal ability was constructed. Each participant's standardised verbal ability score was subtracted from their standardised spatial ability score, to give a measure (S-V) that was at a maximum for those relatively high in spatial ability and at a minimum for those relatively high in verbal ability (after Cronbach and Snow, 1977). The difference between S-V scores for all possible "virtual" checker / tester pairs was computed, and correlated with the number of faults detected by either one or both members of the virtual pair ($C \cup T$). A big difference between S-V scores would indicate a diverse pair in terms of their relative spatial/ verbal abilities. However, no significant association was found between this measure and fault detection performance of the pair.

The mean proportion of faults detected in each of the experimental cells is shown in Table 13.

Error detection performance was analysed using a 2 (task type - checking, testing) x 4 (fault type - ASP, ORD, ORS, ROU) x 2 (CO - commission, omission) ANOVA. Task type was a between participants measure, the fault type and CO factors were within participants measures.

Table 13: Proportion of faults detected by task type, fault type and omission / commission

Fault type	Checking (n = 12)		Testing (n = 14)	
	Omission	Commission	Omission	Commission
	M (*SD*)	*M* (*SD*)	*M* (*SD*)	*M* (*SD*)
ASP	.96 (.14)	.92 (.29)	.83 (.39)	.42 (.51)
ORD	.67 (.49)	.71 (.37)	1.00 (.00)	1.00 (.00)
ORS	.92 (.29)	.75 (.45)	.92 (.29)	.88 (.31)
ROU	.96 (.14)	.83 (.39)	.88 (.31)	1.00 (.00)

There was no significant main effect of task type ($F [1, 24] < 1$) or fault type ($F [3, 72] = 2.08$, $p > .05$). Errors of commission were detected significantly less well than errors of omission ($F [1, 24] = 10.01$, $p < .01$). There was a highly significant interaction between task type and fault type ($F [3, 72] = 10.58$, $p < .001$). This interaction seems attributable to poor performance of checkers in detecting ORD faults, and poor performance of testers in finding ASP faults. This effect was clarified by a significant third order interaction between task type, fault type and CO ($F [3, 72] = 5.59$, $p < .005$). This was attributable to the low number of ASP faults of commission detected by testers. This result also probably accounted for the significant second order interaction found between fault type and CO ($F(3,72) = 4.32$, $p < .01$), where ASP faults of commission were detected poorly compared to faults of omission.

7.3 Discussion

The results must first be considered in relation to the conditions that must be met before diversity could play a beneficial role in fault detection (see § 6.1. for details).

First, there was consistency to error, as evidenced by variation in performance being associated with different fault types across task conditions. Second, neither of the task conditions was a subset of the other in terms of fault detection performance. Also, the task was sufficiently complex that perfect performance was not achieved, as evidenced by the lack of 100% fault detection. In fact, the two conditions were not even found to differ significantly from each other in terms of overall faults detected.

The first of the above points highlights the main finding; that there was diversity of fault detection across task environments. Checkers detected fewer ORD (Opposing Route - Different points position) faults than testers. Testers were comparatively poor at detecting ASP (ASPect control) faults, and this was particularly the case for faults of commission. These differences in performance can be understood if considered in the context of the mechanics of the actual fault finding process that checkers and testers must go through.

To detect ORD faults, the checker must first work out which routes are opposing to each other from the signalling diagram. Then, the points positions for each of these opposing routes must be established. When this is done, the direction that trains would travel as they cross the points, for those routes that must not be set together, must be worked out and translated into a machine readable SSI sub-route label (see § 5). This sub-route labelling has been found to be a comparatively error-prone procedure, as it does not consistently produce the same label for the same spatial direction of route for different track layouts. Only when these activities have been carried out can the checker assess whether the code shown on screen is correct or not.

On the other hand, the tester has a much more straightforward task to find the same fault. They must also first work out which routes are opposing to each other, but then they merely have to attempt to set the two opposing routes at the same time by clicking on the two route buttons. If they can be set together, as shown by an indication on the screen turning from red to green, there is a fault. This result replicated a finding from the error analysis of the DPP (detailed in § 3.2.2), where checkers were again poorer than testers at detecting opposing locking faults.

In contrast to the above, checkers are superior in detecting ASP faults of commission. To the checker these faults would be reasonably obvious. In the SSI code for a particular route, there would appear a reference to a section of track that was irrelevant to the route in question. For the tester to discover this fault, however, would involve them repetitively searching through all the track sections in the layout, whether part of the route under test or not, to see if any affected the functioning of the route under test. This finding is contrary to that found in the field studies. In the fault log audit, testers were found to be superior to checkers for ASP faults. This difference could be because of the nature of the simulation. For instance, because of the restricted functionality of the SSI simulator, it was not possible to include more than eight faults per layout. Any more and it would have become impossible to test due to the lack of working functions. This meant that the individual faults seeded into the simulation were necessarily only a fraction of the variety that can occur in the actual DPP. It was possible that those particular faults chosen for inclusion, while taken from examples in the fault log audit, were not entirely representative.

Cognitive abilities do not appear to have been related to fault finding in this experiment. There were no significant associations found between measures of spatial or verbal ability and the type of faults found. However, there were not even any significant correlations found between ability and overall fault detection performance. Spatial and verbal ability have been shown in the past to be strongly related to performance in computer-based tasks. It was possible that a number of factors to do with the sample may have been having an effect. First, the experiment was somewhat exploratory, and so the sample size was small. Second, the sample had a significantly higher mean and lower standard deviation for cognitive ability than that found in a reference sample of the general population. This was equivalent to a mean IQ of 120 (SD = 15) for the participants in this study compared to an IQ of 100 (σ = 20) for a normative sample of the U.S. adult population. However, it could also be argued that people employed in the 'real world' task of safety-critical software validation would also be highly selected engineers.

Overall, the data tentatively support the findings from the field studies. They indicate that the use of diversity in terms of task environment is an achievable method of promoting diversity in fault detection performance in the SSI design task. However, it is not clear how the various factors that go together to make up task diversity contributed to the results. As discussed earlier, factors such as differences in information saliency, modality of presentation of information etc., are the most obvious dimensions of contrast between checking and testing. However, other, less directly related factors necessarily go hand-in-hand with task diversity. For instance, the domain knowledge presented to checkers and testers was somewhat different. This was unavoidable because of their different tasks, and the differences were controlled where possible with common training. However, the variation that remained may have led to systematic differences between checkers and testers in terms of their mental model of even the higher-level SSI system. Also, the natural variation between individuals' mental models within tasks may be found to be a useful source of diversity.

This factor, in particular, will be investigated further in the next chapter.

8 Cognitive Diversity Experiment II

Experiment II was planned to investigate the role that differences in participants' mental models played in diversity of fault detection. In experiment I it was found that task environment was strongly associated with the types, but not numbers, of faults found by participants in the simulated SSI task. The second experiment would probe the role that variations in participants' mental models played upon diversity in fault detection.

Differences in mental models have been studied in computer programming environments, usually in relation to the differences between novice and expert programmers. Adelson (1981) found differences in the recall of previously presented programme code by groups of programmers with different levels of experience. Expert programmers' recollections tended to be organised semantically, novice programmers' syntactically. Cooke and Schvaneveldt (1988) also found differences between expert and novice programmers in terms of relatedness-ratings of programming concepts, with experts tending to mis-define concepts less often than novices. Although these differences between expert and novice mental models have been used to account for the usually superior programming and debugging performance of experts, this has not always been shown to be the case. For instance, Adelson (1984) found that experts performed better when presented with materials encouraging an abstract representation of a programming task. However, when materials encouraging a concrete representation were presented the performance of novices was better.

These studies show that although differences in mental models can lead to differences in performance the reasons for this are not only because some mental representations are better or worse than others, but also because in different contexts some are more or less appropriate than others. The differences between models can thus be exploited as a dimension of cognitive diversity.

The difficulty with this approach lies in achieving a quantitative assessment of a mental model. One possible method is to use psychometric measures of conceptual distance with respect to concepts within the task environment (see Cook, 1994 for a review).

Although little work has been done in this area, there are some encouraging indications of the validity of this method (Coury, Weiland and Cuqlock-Knopp, 1992; Pallant, Timmer and McRae, 1996).

8.1 Method

Twenty four participants (four female) were recruited from the student population at the same UK University that hosted experiment one. All participants were enrolled on science-based courses. Mean age was 25 yrs (SD = 3 yrs). Participants were assigned randomly to either the checking condition or the testing condition, with the constraint that each condition should have 12 participants.

Experiment II was conducted in essentially the same manner as experiment I, but with the following difference. Instead of completing measures of spatial and verbal ability, after the period of 'core' training (but before task-specific training) participants completed psychometric measures designed to evaluate their mental models of the task environment. Eight constructs were chosen to represent the task environment. These comprised the four components of the railway, combined with the binary states that each could assume, as shown in Table 14.

Table 14: Railway signalling constructs used in psychometric mental model assessment

Construct	Component	Possible states
1.	Points	a. Free to move
2.		b. Locked
3.	Route	a. Not set
4.		b. Set
5.	Signal	a. Green
6.		b. Red
7.	Track section	a. Clear
8.		b. Occupied

Three questionnaires were administered in total. The first was an assessment of the conceptual distance between all possible pairings of constructs. For each pairing, participants were asked to mark on a 0 - 100 scale the 'distance' they perceived between the constructs, zero being lowest possible distance, 100 the highest.

The second questionnaire required participants to rate the relative contribution that the constructs made to the safe running of the railway. For each possible pairing of constructs a 100 - 0 - 100 scale was shown, with the members of the pair as scale anchors. Participants were asked to rate which of the two constructs contributed most to the safety of the railway, or to give a rating towards the middle (0) of the scale if they adjudged the two constructs to be equally important for safety.

The third questionnaire was identical to the second, except that it required participants to evaluate the constructs in relation to the concept of "functionality"; the efficient running or 'liveness' of the railway. All mental model questionnaires are shown in appendix C.

8.2 Results

The mean proportion of faults detected in each of the experimental cells is shown in Table 15.

Fault detection performance was again analysed using a 2 (task type - checking, testing) x 4 (fault type - ASP, ORD, ORS, ROU) x 2 (CO - commission, omission) ANOVA, the latter two factors being within-participants measures.

There was found to be no main effect of task type, or of CO. There was a significant main effect of fault type, however ($F [3, 66] = 8.37$, $p < .001$), with both ORD and ORS opposing route faults being detected poorly and ROU faults detected best. This difference in fault detection performance was highlighted by a significant second order interaction between task type and fault type ($F [3, 66] = 4.94$, $p < .005$). This effect seemed due to testers being less effective in detecting ASP faults, and checkers less effective in detecting ORS faults.

Table 15: Proportion of faults detected by task type, fault type and omission / commission.

Fault type	Checking (n = 12)		Testing (n = 12)	
	Omission	**Commission**	**Omission**	**Commission**
	M **(*SD*)**	***M*** **(*SD*)**	***M*** **(*SD*)**	***M*** **(*SD*)**
ASP	.75 (.45)	.75 (.45)	.71 (.43)	.25 (.45)
ORD	.42 (.51)	.58 (.51)	.58 (.51)	.38 (.48)
ORS	.33 (.49)	.25 (.45)	.46 (.50)	.58 (.51)
ROU	.88 (.31)	.75 (.45)	.67 (.49)	.67 (.49)

These effects were further clarified by a significant third-order interaction between task type, fault type and CO ($F [3, 66] = 4.62$, $p < .01$). This is attributable to testers showing a great disparity between errors of omission and commission in ASP faults, with errors of commission detected particularly poorly.

8.2.1 Analysis of mental model data

The distance between constructs from the three mental model questionnaires were analysed separately using the INDSCAL Multi-Dimensional Scaling (MDS) procedure (as described in Young and Harris, 1992). Each participant's responses to each questionnaire were assumed to represent ordinal rather than interval- or ratio-level data, therefore requiring non-metric MDS. Participants' responses were made to all possible pairs of constructs; therefore the data were represented as square, symmetrical matrices. For each questionnaire the matrices from all participants were aggregated to produce one overall model based on weighted Euclidean distances.

A scree-plot of S-stress against dimensionality suggested a two-dimensional solution for each questionnaire. Inspection of the weirdness index for each participant showed

no values greater than .8. Particularly high or low weirdness values were not found predominantly from either the checkers or testers, indicating that the MDS solution was acceptable for both groups. Overall RSQ values for each questionnaire were not high (all between .50 and .60), but S-stress values were acceptable (< .3), and lower than for 1- or 3-dimensional solutions.

With a two dimensional model, one flattened weight was produced per participant for each of the conceptual distance, safety, and functionality questionnaires. The flattened weights provided an index of an individual's deviation from the mean solution for the entire sample, and therefore an index of how similar or different the participants' mental models were from the average.

The flattened weight from each questionnaire was correlated with the number of faults detected. No significant associations were found, showing no straightforward association between individual performance and mental model "deviation".

To gauge the effects of diversity of mental models on error detection, 'virtual' pairs of participants were constructed. All possible different combinations of participants were first listed; this is equal to [N * N – 1] / 2, or 276 virtual pairs in this case. For each virtual pair the difference between flattened weights was computed for each of the questionnaires.

A number of different measures of fault detection performance could have been taken for these pairs. The mean of two participant's individual fault detection scores gives an indication of the overall merit of the pair, but does not include any information as to the levels of diversity or redundancy in the pair. A better indication of the practical benefit of task diversity is to look at the total number of faults detected by pairs corresponding to C ∪ T in Figure 6. This corresponds to the total number of faults detected by both members of the pair, and so to their overall effectiveness.

This "effectiveness" fault detection variable was then correlated with each of the mental model difference variables. Only the difference in the 'functionality' mental

model scores was found to be significantly related to fault detection performance ($r[276] = .24, p < .001$).

8.3 Discussion

Overall, the pattern of performance found in this experiment was similar to that found in experiment I. Testers were again found to be relatively poor at detecting ASP faults of commission, and checkers were again found to be poor in detecting opposing locking faults, but this time for routes set over the same points position rather than different points position. While this finding is slightly different from that found in experiment I, it replicates the pattern found in the field studies, and relates once again to the use of sub-routes.

In general error detection was poorer in experiment II than experiment I. This could be attributable to the less stringent sampling criterion. Although the participants came from a similar population in terms of educational level, pragmatic considerations meant that sampling could not be constrained to only engineers and computer scientists. The resulting participants would therefore not necessarily have had experience of code inspection and functional testing before the experiment.

Considering the analysis of mental model data, the results showed no relationship between mental model measures and absolute performance. This suggests that the participants' mental models were not significantly 'better' or 'worse' but merely different.

The diversity measure between 'virtual' pairs of participants showed only one significant relationship, between diversity with respect to concepts of functionality and overall fault detection effectiveness of the pair. There are a number of speculations that can be made about this finding.

On the one hand, what constitutes the functionality or liveness of a system may be a more fluid notion than that of safety, and therefore admit more variation in its conception. Safety and liveness are concepts central to the operation of systems that have similar properties to the railway considered here. For instance, distributed

computer systems have to move multiple packets of data (analogous to trains) across fixed wiring networks (analogous to rail networks) as efficiently as possible without losing data. It has been demonstrated in this domain that proving the existence of safety properties of networks is an easier job than doing the same for liveness (Alpern and Schneider, 1987), in part because of the greater ease with which safety can be defined.

To illustrate this point, think about trying to evaluate the state of a railway for its safety and liveness. For the purposes of this example I will define 'railway' quite simply: just two railway lines (A and B) that cross each other, with two sets of signals (A and B, one on each line protecting the junction) and two trains (A and B), one on each line. If train A is travelling along its line towards the crossing point, then signal B (on the other line) should be red. If train B were obeying signal B, and had stopped before it, then the railway would certainly seem to be within its envelope of safe operation. However, is the system within its liveness envelope? On the one hand, train B is stationary and that doesn't seem very 'live'. However, this delay is surely a necessary part of a safe system, and will only conflict with liveness goals if signal B continues to show a red light long after train A is clear of danger. Therefore, a red signal is always good from a safety point of view, but sometimes acceptable and sometimes not from a liveness point of view.

However, if the degree of variation possible within a concept is the important factor, then 'conceptual distance' should have also been related to error detection. Unlike the safety and functionality/liveness questionnaires, the participants were not given any guidance as to the goal that any similarity or difference between railway elements should serve with the conceptual distance questionnaire. Thus, conceptual distance should perhaps have accommodated even more variation between participants' mental models than did functionality.

It could be argued, however, that it was precisely the focus on task relevant factors given by the safety and functionality questionnaires that was important. Participants might have used many task-irrelevant factors to decide their answers on the conceptual distance questionnaire, e.g. how big the railway elements are.

This point is unlikely to be resolved here by debate based on such limited data. Experiment III would provide a further test of the exploitability of mental models in the service of diverse task performance.

9 Cognitive diversity experiment III

Of the mental model measures, only differences relating to the functionality of the railway were found to be associated with improved fault-finding performance in Experiment II. This indicated that if people have different models of what constitutes efficient working of the railway, they will tend to find different faults. Functionality corresponds to flexibility or 'liveness' of the network, where points are free to be moved and routes can be set without undue restriction. This perhaps allowed more diversity between people's mental models than the more constrained and perhaps easier to define concept of safety. However, the conceptual distance measure should be the most freely definable of them all, supporting large differences in mental models. Perhaps in the case of conceptual distance, the range in mental models was too diverse, leading to no predictable association with performance. 'Conceptual distance' alone may be too sprawling a notion to have any useful impact.

In experiment II the psychometric mental model measures were administered after the core training in railway principles, but before the specific training for checking or testing. This meant that the variation found between participants' mental models would have represented natural variations between their understanding and conceptualisation of the training, without any systematic effect of the different tasks they were to perform (i.e. checking vs. testing) or the variations in domain knowledge that the specific training would bring. The large and consistent interaction between fault- and task-type found in experiments I and II may also have been due in part to the influence of 'environment-sponsored' differences in the range of mental models that each task supported, or that fit 'best'. Any environment-sponsored differences in mental model would play a mediating role in the type of faults found by participants in each condition.

The dimension of safety vs. functionality showed some promise in experiment II. Safety and functionality were used in this experiment to investigate the effects that task environment *per se,* and the influence of task environment upon participants' mental models, has upon fault detection performance. Therefore, the faults seeded into experiment III differed in terms of whether a fault affected the safety or functionality

of the railway network. It would then be possible to investigate whether the task environment is associated with the numbers of safety and functionality faults detected. Participants would also be asked to search specifically for either faults of safety or faults of functionality. If their mental model of the safety or functionality of the railway is psychologically meaningful, over and above the effect of the environment itself, then this should be evident in their ability to find their assigned faults.

9.1 Method

9.1.1 Participants

Eighty-eight participants (9 female) with a science background were recruited from the student populations at four UK universities and two sixth-form colleges. Mean age was 20 years, (SD = 3 yrs). Participants were randomly assigned to conditions; thirty nine participants were assigned to the checking condition, 49 to the testing condition. (Numbers are unequal because of the unforeseen cancellation of an experimental session.)

Experiment III was then conducted in a similar manner as experiments one and two, but with the following differences.

9.1.2 Fault categories

In the previous experiments the sixteen faults seeded into the SSI simulator were made up of two each from four signalling categories: ASP, ORD, ORS and ROU. Each category contained one error of omission and one of commission. In this experiment it was still necessary to have the four signalling categories, but this time a completely new set of sixteen faults was chosen so that each of the categories contained one fault affecting the safety of the railway and one affecting the functionality. To subdivide the categories any further would not be possible with only sixteen faults, so the commission/omission dimension was controlled for by having only faults of commission. (Having more than sixteen faults would have made the testers' task unworkable, as different faults would start to 'hide' each other). Training was slightly altered to emphasise the distinction between the safety and functionality of the railway, and the practice session included faults of both types. The new within-participants factor of safety vs. functionality was named S_F.

9.1.3 Procedure

The experiment was initially conducted as per experiment I. All participants completed tests of verbal and spatial ability taken from the General Aptitude Test Battery (US Dept. of Labor, 1982). This was followed by both core and specific training sessions. Participants then completed the three psychometric mental model measures used in experiment II. Note, in this experiment, as opposed to experiment II, the psychometric mental model measures were presented to participants after they had completed both their core (general) and specific (checking or testing) training. This was to emphasise the potential task environment-mediated differences between the mental models of checkers vs. testers.

Before the participants proceeded with the fault finding task they were allocated randomly into another two groups. Approximately half in each condition were instructed to concentrate 80% of their effort on finding faults of functionality (and 20% on safety), the other half were told to concentrate 80% of their effort on finding faults of safety (and 20% on functionality). This factor was referred to as INST (INSTructions).

A time limit was this time imposed on task performance, amounting to approximately 75% of the mean time that participants in the previous experiments took before finding their final error. For layout one (with seven routes) this corresponded to 30 minutes; for layout two (with nine routes) this was 40 minutes. (Participants were free to finish their task before the time limit.) It was felt that if the task was self-paced, as in previous experiments, participants might be able to detect too high a proportion of the errors. This ceiling effect would mask distinctions between their efforts to find errors of safety vs. functionality and vice versa.

Participants then proceeded to complete the task. After they had finished both layouts the mental model questionnaires were completed for a second time. Three additional questionnaires were then administered. The Error Orientation Questionnaire (EOQ; Rybowiak, Frese, Garst & Batinik, 1999); eight scales designed to elicit participants' self-rated attitudes to their own errors. The Cognitive Style Index (CSI; Allinson and Hayes, 1996) contains one bipolar scale of self-rated cognitive style. A low

score indicates a global, holistic problem-solving approach; a high score indicates an analytic approach. Finally a personality assessment questionnaire was completed. Prevue ICES (Bartram, 1994) contains four scales; Independence, Conscientiousness, Extroversion and Stability, each made up from two minor scales.

9.2 Results

Three participant's data were removed from the analysis: two testers detected no faults at all, and there was some doubt as to whether they had fully understood the final task instructions; one checker detected all of the faults in both layouts (which, for this individual, meant that perfect performance seemed to be reliably obtainable, obviating the need for diversity [see § 6.1]). Thus, 85 participants' data were retained for analysis.

9.2.1 Individual differences

There were no significant differences between groups in terms of spatial or verbal ability. Both spatial ability ($r\ [85] = .35$, $p < .005$) and verbal ability ($r\ [85] = .51$, $p < .001$) were found to have significant positive correlations with the overall number of faults detected. There were found to be no significant correlations between fault detection performance and analytic/global cognitive style, or any of the personality scales. There was a marginally non-significant negative trend between fault detection and the EOQ scale of covering-up errors ($r\ [85] = -.21$, $p = .053$), so that poorer fault detection performance was associated with individuals who report that they tend to hide their errors. No other EOQ scales were significantly associated with performance.

There was a significant correlation between the number of faults detected and whether English was the participant's first language or not ($r_{pb}\ [85] = .27$, $p < .05$), such that those with English as their first language tended to find more faults. This language variable is also confounded with checking and testing groups, such that the testing group contained more participants whose first language was not English than did the checking group ($t\ [68] = 3.87$, $p < .01$, with Welch's correction because of unequal variances between groups). To control for this potentially biasing factor, whether the participant's first language was English or not was used as a covariate in subsequent analyses.

9.2.2 Fault detection performance by treatment group

The data were analysed using a 4 (fault type) x 2 (S_F) x 2 (task type) x 2 (INST: instructions to search for either safety or functionality faults) ANOVA, with 'English is First Language' (EFL) as a covariate. The first two factors were within participants, the latter ones between participants.

There were found to be no significant effect of INST ($F[1,80] < 1$), so the analysis was repeated without it. Table 16 shows the proportions of faults detected by task type, fault type and S_F.

Table 16: Proportion of faults detected by task type, fault type and safety vs. functionality

Fault type	Checking (n = 47)		Testing (n = 38)	
	Safety	Functionality	Safety	Functionality
	M (*SD*)	*M* (*SD*)	*M* (*SD*)	*M* (*SD*)
ASP	.58 (.32)	.70 (.36)	.67 (.38)	.22 (.37)
ORD	.44 (.39)	.79 (.30)	.72 (.37)	.26 (.31)
ORS	.27 (.33)	.54 (.39)	.62 (.37)	.49 (.45)
ROU	.86 (.29)	.69 (.38)	.85 (.26)	.26 (.38)

In this analysis the overall effect of EFL was marginally non-significant ($F [1, 82] = 3.84$, $p = .052$), but it was decided to retain it as a covariate as it was found to be significantly associated with poorer performance on ASP and ROU safety faults.

There were no main effects of task type ($F[1, 82] = 1.37$, $p > .05$), fault type ($F[3, 246] = 2.44$, $p = .065$), or S_F ($F[1, 82] < 1$). As found previously there was a significant

two-way interaction between task type and fault type ($F [3, 246] = 10.22$, $p < .001$), most notably because of the checkers' somewhat superior performance in detecting ASP and ROU faults, and their poorer performance in the detection of ORS faults.

There was a significant interaction between task type and S_F ($F [1, 82] = 163.25$, $p < .001$), with checkers detecting more functionality faults compared to safety faults, and vice versa for testers.

Finally, there was a significant three-way interaction between task type, fault type and S_F ($F [3, 246] = 3.23$, $p < .05$). For ASP, ORD and ORS fault types, checkers detected more faults of functionality, testers more faults of safety. But for ROU faults, checkers detected more faults of safety than of functionality.

9.2.3 Analysis of diversity in performance

To investigate diversity in fault detection, as before all possible virtual combinations of pairs of participants were constructed. Previously, the total number of faults detected by the pairs, corresponding to $C \cup T$ in Figure 6, was used as the measure of fault-finding effectiveness. However, this measure does not give an indication of the amount of diversity in fault detection for the pair, and so their resistance to common mode error. A measure of diversity is given by the number of faults detected by only one or other member of the pair, but not both, and corresponds to the area of diversity, $(C \cap T') \cup (C' \cap T)$ in Figure 6. The data were also broken down by pair type; the virtual pairs of participants consisted of checkers with checkers ($n = 703$), testers with testers ($n = 1081$), or checkers with testers ($n = 1786$). The mean proportion of faults detected, for each fault measure and by pair type, is shown in Table 17.

For the mean faults measure there was a significant difference between pair types ($F [3, 3567] = 86.26$, $p < .001$). Post-hoc Bonferroni multiple comparison tests showed that each group was significantly different to the others with pairs of checkers having the highest mean score and pairs of testers the lowest.

Table 17: Proportion of faults detected by measure of fault detection and 'virtual' pair type

Measure of fault detection	'Virtual' pair type		
	Checker and Checker	**Tester and Tester**	**Checker and Tester**
	***M* (*SD*)**	***M* (*SD*)**	***M* (*SD*)**
Mean of pair	.61 (.15)	.51 (.15)	.56 (.16)
Redundancy and diversity C ∪ T	.80 (.14)	.71 (.17)	.82 (.16)
Diversity (C ∩ T') ∪ (C' ∩ T)	.40 (.15)	.40 (.14)	.52 (.14)

Note: n(checker and checker) = 703; n(tester and tester) = 1081; n(checker and tester) = 1786

When considering the C ∪ T measure, there was again found to be a significant difference between pair types ($F [3, 3567] = 117.13$, $p < .001$). Post-hoc tests showed that pairs of testers were detecting significantly fewer faults than the other two pair types, but checker with checker pairs were not significantly worse than checker with tester pairs.

The difference between groups for the diversity measure [(C ∩ T') ∪ (C' ∩ T)] was highly significant ($F [3, 3567] = 349.56$, $p < .001$). Post hoc analysis showed pairs containing a checker and a tester detected significantly more diverse faults than the pairs containing either only checkers or only testers.

9.2.4 Diversity of Individual differences

It was hypothesised that differences in cognitive style, cognitive ability and personality would lead to differences in the types of faults detected, rather than merely the number. To investigate this all the possible unique pairings of participants were constructed as before. The absolute difference between the two members of the pair

was then taken for the above mentioned variables. Correlations between the difference measures and the measure of absolute fault detection performance of the pair (C ∪ T) and diversity performance [(C ∩ T') ∪ (C' ∩ T)] were then calculated. Because of the large number of pairs in this analysis (N = 3570) a correlation of only slightly more than .03 would be significant at the .05 level. Because of this and the large number of correlations being performed, only correlations with an absolute value above .10 are reported to avoid too many seemingly significant but spurious relationships.

None of the differences for the cognitive ability variables, personality variables or the cognitive style index were found to have correlations of greater than .10 with fault detection. Regarding the EOQ scale, differences in 'error risk taking' ($r[3570] = .12$, $p < .001$) and 'thinking about errors' ($r[3570] = .13$, $p < .001$) were found to correlate with overall fault finding performance (C ∪ T).

9.2.5 Mental models

As for experiment II, the distance between constructs from the three mental model questionnaires were analysed separately using the INDSCAL multidimensional analysis procedure. Parsimonious solutions for each questionnaire were derived, again seeking to minimise S-stress while maximising the proportion of variance accounted for by the solution. For the conceptual distance measure this was a three dimensional model accounting for 49% of the variance in the original distance matrix. A four dimensional solution accounting for 56% of the variance was selected for the Functionality measure. Finally, a two dimensional solution (43% of the variance) was chosen for the safety questionnaire.

For each questionnaire a number of flattened weights were derived, equal to the number of dimensions in the solution minus one. These weights gave an index of the difference between each participant and the mean for the entire sample. For each of the possible pairs of participants, the absolute difference between each of the participant's flattened weights was taken as a measure of dissimilarity of the pair. These difference measures where then used in a regression analysis to predict the C ∪ T fault detection performance measure for the pair (Table 18).

Table 18: Results of regression analysis of mental model differences on fault detection performance

Predictor		R	R^2	F [2, 3567]	p <	β	t [3567]	p <
Conceptual distance		.06	.00	6.38	.005			
	W1					.03	1.96	.051
	W2					-.06	3.42	.001
Functionality		.16	.03	30.47	.001			
	W1					.14	8.03	.001
	W2					-.09	5.56	.001
	W3					-.07	4.31	.001
Safety		.08	.01	22.94	.001			
	W1					-.08	4.79	.001

Note: W1, W2, etc. = Multidimensional scaling flattened weight

All three sets of flattened weights were significantly associated with fault detection performance, as shown in Table 18 . For all of the mental model difference measures the association with fault detection performance was lower than that found for experiment II; only the functionality measure had a multiple correlation coefficient of greater than .10. Of the individual β weights for the functionality measure only the most important contributor, weight 1, is positively associated with fault detection; the others are associated negatively with performance.

The relationship between individual weights and fault detection was further investigated by dividing the sample into three equal groups for each of the three functionality weights. Each pair of participants was thus a member of one of three groups for each of the weight measures: low diversity, medium diversity and high diversity. Mean fault detection performance was then analysed by diversity group using one-way ANOVAs with post hoc Bonferroni multiple comparisons. For weight one, the high diversity group detected significantly more faults than either the middle or low diversity groups ($F [2, 3567] = 17.20, p < .001$). For weight 2, the high diversity group

this time detected significantly fewer faults than either of the other two groups ($F [2, 3567] = 10.20$, $p < .001$). Weight three showed evidence of a quadratic relationship between diversity and fault detection. The middle diversity group this time detected significantly more faults than the either the low or high diversity groups ($F [2, 3567] = 20.52$, $p < .001$).

9.3 Discussion

Although experiment III used a different set of faults to that used in the experiments I and II, there were similar patterns found in fault finding performance across tasks. Again, the worst fault detection performance for testers was for ASP faults, and for checkers it was ORS faults (involving sub-routes).

There was a significant relationship between task-type and whether the faults found were predominately ones of safety or of functionality/liveness. Checkers tended to be better at detecting faults affecting safety rules whereas testers tended to be better at finding faults that affected only functionality.

The reason why testing should be better for safety faults and checking better for functionality faults is not wholly apparent. It does not seem to be due in any great part to matters of individual effort or goal preference, as the manipulation of task INSTructions (where one half of participants were asked to predominantly search for safety faults and vice versa) was not found to have any significant effect. This is so despite the experiment having adequate statistical power to detect 'moderate' effects. (Experiment III had a power of .78 to detect 'moderate' differences [e.g. Cohen's $f > .3$] between INST groups [with $\alpha = .05$, two-tailed]).

The relation of safety vs. functionality efficacy between checking and testing is in agreement with findings from the error analysis (§ 3.5.1), however. There, it was argued that being able to see the all of the component parts of the code allowed checkers to notice additions what were clearly not part of the route being evaluated, and which usually specified additional restrictions on the liveness of the railway. To find the same fault testers would have to laboriously evaluate all of the elements of the railway that were not part of the route under test.

The only individual difference measures found to have substantial relationships with individual fault-detection performance were spatial and verbal ability. This is in agreement with previous literature on programming skill (see van der Veer, 1989, for a review), although in this experiment verbal ability was found to have a stronger relationship with fault-finding performance than did spatial ability, whereas the reverse has been found to be more often the case.

The effect of spatial and verbal ability on fault finding did not emerge in the first experiment, however, probably because of the more highly selected nature of the sample in that case. In the first experiment participants were aged around 24 years, a mix of post- and undergraduate students, and all had backgrounds in engineering or computer science. Participants in the third experiment were younger, around 20 years on average, and were predominately undergraduates or final-year sixth-form students with 'only' a science background.

None of the Cognitive Style, Personality, or Error Orientation measures was found to be significantly related to individual fault detection performance.

9.3.1 Diversity

Differences between virtual pairs of participants in terms of their individual difference measures scores were not found to be substantially related to any measure of fault detection. The only measures that did correlate above a level of .10 were the EOQ scales of 'error risk taking' ($r = .12$) and 'thinking about errors' ($r = .13$). Another EOQ scale, 'covering-up errors' was found to be marginally non-significantly related to individual fault detection ($r = -.21$). While these results seem connected by virtue of their parent questionnaire (the EOQ), they are all separate scales, and show, at best, weak relationships with fault detection performance. For example, the differences in EOQ scales accounted for little more than 1% of the variation in virtual pair fault detection performance.

Regarding the psychometric mental model measures, a very mixed pattern of results was obtained. On the one hand, very different Multi-Dimensional Scaling solutions were found between experiments II and III. In experiment II, two-dimensional

solutions were arrived at for conceptual distance, safety and functionality. In experiment III, three-, four- and two-dimensional models were found, respectively. This could indicate that the changes made to the administration of the mental model scales between experiments II and II had some effect. The extra, task-relevant training that the participants completed in experiment III compared to experiment II could have resulted in more complex mental models in experiment III, as demonstrated by the more complex MDS solutions. However, the solutions for experiment III accounted for slightly *less* of the variance in the raw responses than did the more 'complex' experiment III solutions.

On the other hand, it was found consistently in experiments II and II that variation in conceptions of the functionality or liveness of the railway was most importantly related to fault-finding performance, and not variations in the concept of safety (or of conceptual distance).

However, in both experiments the amount of variation in fault detection scores accounted for by the functionality measure was very small; 6% and 3% respectively. Further, of the three functionality-related flattened MDS weights used in the regression analysis in experiment III, only one was actually related positively with error detection performance. This is particularly problematic because the 'meaning' of the various dimensions in the MDS solutions is difficult to determine (unlike factor analysis methods, where the meaning of the individual items can be abstracted to latent variables).

In summary, it seems that there is perhaps some promise in using individual differences as a domain for cognitive diversity. However, the data presented suggest that it is individual's conceptions of the task, rather than their abilities or styles, which may be most important. A first step in investigating this issue further might be to use a more established technique in eliciting mental models, e.g. Repertory Grids (Kelly, 1955). This technique in particular would allow the participant to produce their own conceptions of the important elements in the task space, as well as the relationships between elements, in an interpretable manner.

The only factor that was strongly related to diversity in fault detection performance was the task-type: checking or testing. This is a confirmation of the findings from the first two diversity experiments. However, is this diversity a useful factor in promoting more robust fault detection?

Inspection of Table 17 shows that, when considering the mean fault detection performance of pairs of checkers (CC), testers (TT), and checker/tester pairs (CT), CC pairs were the most effective, finding 61% of faults on average. CC pairs were significantly better than CT pairs (56%), who were also better than TT pairs (51%).

The mean fault detection figures for CC and TT pairs are merely the mean number of faults found by individual checkers and testers, respectively. (This can be confirmed by taking the mean of the checkers' and testers' scores from Table 16 and comparing them with the mean detection performance figures in Table 17. The figure for CT pairs is just the mean of the individual checkers' and testers' performances [CT = 56% = (61% + 51%) / 2]. The mean detection performance of virtual pairs is thus just the mean of individual performances. When taken as N = 85 individuals this difference [61% for checkers, 51% for testers] was not significant. However, the same difference was launched into statistical significance when considered as N = 3570 pairs; however, this is not valid statistically, as the 3570 pairs are not independent.)

If there existed no other information about fault detection performance across checking and testing, the superiority of checking would point to abandoning the testing task and merely having two checking phases. Any use of the 'weaker' testing task would seem to just be diluting the performance of the checkers. However, this position ignores the influence of common-mode error, as will be shown below.

Checking was found, on average, to detect 61% of faults, and therefore leave 39% undetected. Therefore, we might assume that a second check of the same code would, on average, detect 61% of the remaining 39% of faults, giving a grand total of 85% of errors detected after two checks (39% x .61 = 23.79%. 61% + 23.79% = 84.79%, rounded to 85%. An easier way to compute the same thing would be to consider the proportion of faults missed: checkers found .39 on average, so a second application

would miss .39 of the original .39, i.e. $.39^2 = .15$. 1 - .15 = .85 of faults found). However, if we look at the total number of faults detected by one or both members of the pair (i.e. C ∪ T in Table 17), the measured figure is only 80% detection on average, 5% less than the 85% figure that might have been expected.

The same pattern is found for TT pairs. Mean fault detection performance of individual testers (and TT pairs) is 51%, so the proportion of faults missed was .49. A second application of testing would therefore be expected to miss the proportion.49^2 = .24. This is .76 or 76% of faults detected. However, the C ∪ T measure of pair performance was only 71%, again 5% less than might have been expected.

There seems to be a pattern whereby paired performance of checkers with checkers and testers with testers is less effective than might be anticipated when considering individual performance. This pattern is reversed when considering CT pairs.

The average error detection performance of CT pairs, from Table 17, was 56%. (Remember that this is inferior to the mean performance of CC pairs, who managed 61%.) As was pointed out earlier, this figure of 56% is just the mean of the average error detection performance of checkers and testers, who missed 39% and 49% of faults, respectively. Therefore, we might then expect a CT pair to miss on average .39 x .49 = .19, or 19% of faults, finding the other 81%. However, C ∪ T performance for CT pairs is actually slightly better than might have been anticipated, at 82% detection. In fact, CT pair performance (82% detection) is superior to CC pair performance (80%), even though individually checkers outperform testers, on average.

The reason why 'diverse' performance is better is although testers don't find so *many* faults on average as checkers they do tend to find somewhat *different* faults to checkers. This can be seen by inspecting the measure of diversity in performance in Table 17 ((C ∩ T') ∪ (C' ∩ T)). For both CC and TT pairs, only about 40% of the faults detected are unique to either member of the pair. For CT pairs, on average 52% of faults detected by the pair are 'diverse'.

The above findings informally demonstrate the benefit of diversity in fault detection processes. A double application of the same fault detection method, i.e. redundancy, is not likely to be as effective on the second application as it was on the first, over and above the normal law of diminishing returns. This is because, on average, the faults missed by the first application will be 'harder' ones. A second application of the same method will find these faults no less difficult, indeed probably more so because they will be more thinly spread. If the second application is with a diverse fault detection method, however, then some of the faults remaining from the first method might be quite 'easy' ones as far as the second method is concerned, and so be more easily detected.

These conclusions need to be tempered somewhat. The data on which they are based were obtained from a 'toy' programming task, with naïve participants, which only looked at code checking and functional testing tasks. This begs the question as to what are the general properties of task diversity in fault detection, and how can these be reliably measured and used.

9.3.2 Modelling diversity in fault detection

In software design, diversity has been proposed as means of achieving highly reliable systems. Using the metaphor of hardware redundancy (i.e. by arranging components in parallel; § 1.1.2), the practice of N-version computer programming has become widespread in industrial and military safety-critical applications (e.g. Leveson, 1995; Rouquet and Traverse, 1986). N-version programming involves developing two or more separate programmes designed to do the same job. By making the development processes of the different software versions independent it was hoped that the same mistakes would not be made in identical fashion by the different development teams, and therefore the N different versions of software would fail independently of each other. So, although different versions of the software might vary in their individual reliabilities, when used in the same system they should be unlikely to fail upon the same control input or demand, and so together would be likely to ensure greater reliability than any single version.

However, hopes that this initial promise of N-version software could be fulfilled were dashed by a number of experiments which showed that independently-developed software versions tended to fail in correlated fashion, i.e. an input or demand that caused one version to fail was likely to make the other versions fail, too (e.g. Eckhardt et al, 1991; Knight & Leveson, 1986). (Note that it is not implied that a failure in one version would *cause* another version to fail on the same input, merely that there would be a correlation between failures of the different versions.)

The reasons why this should be so were outlined in a model by Eckhardt and Lee (1985). Although different teams developing different software versions could be notionally independent, the difficulty of the real-world problem solved by the software would be the same for all. This notion of 'difficulty' can be used to classify system demands: some operations that the software must manage could be labelled as 'easy', such as the requirement for a signal to turn red when passed by a train. Other operations would be 'hard', such as working out the multiple constraints presented by multiple trains approaching the same track junction. Although independently developed, the N-versions of software would thus be more likely to fail when required to perform a 'hard' operation than when doing an 'easy' one, and they would therefore show correlated (dependent) failure, rather than independent failure.

The Eckhardt and Lee (1985) model of dependent failures was generalised by Littlewood and Miller (1989). They showed that if the different development teams of the N-versions of software were to use processes that differed in how 'easy' or 'hard' different aspects of the programme seemed to them, then the different versions of software produced might even fail in negatively correlated ways. That means that on an input or demand on which version A might be very likely to fail, version B might be very *unlikely* to fail.

Negatively correlated failure is a better scenario than even independent failure between versions. It is akin to the following example for the cautious investor. Imagine that two companies are competing for a large government contract. The stock price of the winner is likely to rise, whereas the price of the loser will probably fall, all other factors being equal. However, you don't know which will win or which will lose, so how do

you invest your money? To be guaranteed of not losing money, you should spread your investment across both companies; if you have £20 to spend then invest £10 in each. If company A wins and their stock doubles, you will have turned £10 into £20. However, company B's stock may have fallen by half, too, reducing your investment there to £5. However, your final sum is still better than you started with, £25 vs. £20, and would be the same if company B won instead of company A.

The important factor in the example above is the negative correlation between the stock prices. As long as one goes up when the other goes down (and by the same ratio) then the initial investment is safe. Similarly, as long as a software design manager can be reasonably sure that they have negatively correlated versions then the pair will always be more reliable than either individual programme.

There are two big problems with the apparent panacea of negatively correlated N-version software, however.

First is the matter of cost. By definition N-version programming is likely to at least duplicate the costs of a single version. However, even though in theory two perfectly negatively correlated software versions may together always be likely to produce the correct output given an arbitrary input, it is not an easy matter to decide which version is actually telling the 'truth'. When hardware fails it is usually fairly obvious, as the component in question often stops working. Failed software may also stop working, but more often it tends to keep working but in a 'faulty' fashion; in effect, it 'lies'. Guaranteeing the detection and correction of "Byzantine Failure" as it has been called, relies upon at least four versions of the software (Shasha & Lazere, 1998). This means that to fully reap the benefits of N-version software at least four completely independent versions of the software have to be produced. Safety-critical software is inherently expensive to develop, but multiplying these costs by at least four is often seen as prohibitively expensive, especially in light of the following, second, pit-fall of N-version programming.

Second, it is not at all clear how to construct the N-different versions of the software so that they will fail in independent or negatively correlated ways; indeed, it is not an

easy task even to *evaluate* the independence of a specific set of diversely developed programmes (Popov & Strigini, 1998). This is because the models of Eckhardt and Lee (1985) and Littlewood and Miller (1989) are purely *conceptual* models of software diversity, and relate to how versions will fail 'on average' rather than in any specific case. Knowledge of specific cases is precisely what is needed in practice, however. For instance, the experiments conducted by Knight and Leveson (1986) found that, although multiple-version software was on average very much more reliable than any individual version, they also found that some of the best single programmes were more reliable than the worst of the multiple-versions.

So, to use the Eckhardt and Lee (1985) and Littlewood and Miller (1989) models in a practical fashion means being able to estimate the key 'difficulty' distribution parameters for an actual set of programmes. However, that would mean having access to data from a large sample of faults from a large sample of programmes to estimate the distribution of 'difficulty' for each programme. Although this has been done in experiments, where many different versions of software were developed (e.g. Nicola and Goyal, 1990), it is not a feasible solution for real safety-critical programming applications.

Further, as the desired reliability of the software becomes greater and greater, as for safety-critical applications, then fault-detection processes become less and less useful for estimating their reliability. This is because as faults become rarer there is correspondingly less and less evidence upon which to make a judgement of reliability. In effect, as the desired reliability of a system increases, the sample size of faults or failures decreases and the confidence limits on those estimates widen.

A way round these problems was put forward by Littlewood, Popov, Strigini and Shryane (2000). They found that if the dependencies between different fault-finding strategies for a *single* software version are modelled, instead of the dependencies between N-different versions, then the task of estimating the required parameters for a given piece of software is tractable.

As in the Eckhardt and Lee (1985) and Littlewood and Miller (1989) models, the Littlewood *et al* (2000) model assumes that different fault-finding tasks vary in how 'difficult' particular faults are to find. This 'difficulty' function is likely to vary across faults in relation to fault finding strategy; some faults being easier to detect with one method, others being easier for the other method. The difficulty of a particular fault for a particular fault-finding method is represented in the model as the probability that a randomly-chosen application of the fault finding method will fail to detect the fault. The mean of these individual probabilities can then be thought of as the ineffectiveness of the fault-finding method as a whole. Finally, the covariance between the ineffectiveness measures of the fault-finding methods is taken as the measure of diversity. The expected fault-finding performance of an application of two fault-finding methods will then be the product of the average ineffectiveness measures plus their covariance, as shown below:

*Overall ineffectiveness = (ineffectiveness of method A * ineffectiveness of method B) + covariance AB.*

Positive correlation between measures will thus add to the ineffectiveness of the overall performance, negative correlation will reduce ineffectiveness, i.e. improve fault-detection.

In the case of experiment III, the proportion of participants not finding a particular fault was taken as the ineffectiveness measure for that fault. Averaging over all 16 faults the overall ineffectiveness of checking was .3947 and testing was .4920, i.e. just the average proportions of faults missed. The covariance between checkers and testers for the 16 individual faults was -.0127. Therefore, the ineffectiveness of a checker-tester (CT) pair would be expected to be:

$$\text{CT ineff.} = (\text{C ineff.} \times \text{T ineff.}) + \text{covCT}$$

$$\text{CT ineff.} = (.3947 \times .4920) + -.0127 = (.1942) + -.0127 = .1815$$

An ineffectiveness of .1815 corresponds to an effectiveness of .8185 (= 1 - .1815), which is the measured C ∪ T overall fault-detection performance reported in Table 17 (.82 rounded to two decimals).

The above figures are based upon the full set of 16 faults in Experiment III. In this case the identity and location of each fault were known beforehand, and using this full fault data allowed the performance between checking and testing to be exactly matched and the precise covariance computed. However, this information will never be normally available in practice (as it is the identity and location of the faults that is being sought!).

Treating each fault as an individual entity means that any fault-detection data collected from actual testers and checkers will most probably not be exactly matched in this way, i.e. they will not discover exactly the same faults. (Indeed, that checkers and testers discover different faults is precisely what is hoped for with diverse methods.)

To circumvent this problem, individual faults can be categorised and grouped into fault-classes. Although individual faults will tend to be 'sparse', by aggregating them into classes enough data can more easily be obtained to allow estimation of the required probabilities and covariances.

For example, in Table 17, rather than present data on all 16 faults from experiment III, the probabilities of detection are aggregated by signalling principle violated and whether the faults were ones of safety or of functionality. In Table 19 and Table 20 the data from experiment III is presented again, this time as *ineffectiveness* figures (1 - effectiveness) broken down by one fault class per table; signalling principle in Table 19 and safety/functionality in Table 20, respectively.

Table 19: Proportion of faults missed (ineffectiveness) in Experiment III by task type and signalling principle violated.

	C	T	CxT
ASP	.3618	.5585	.2021
ORD	.3882	.5160	.2003
ORS	.5987	.4468	.2675
ROU	.2303	.4468	.1029
Mean	.3947	.4920	.1932
		Mean of C x Mean of T =	.1942
		Covariance CT =	-.0010

Note: C = mean proportion of faults missed by checkers; T = mean proportion of faults missed by testers.

Table 20: Proportion of faults missed (ineffectiveness) in Experiment III by task type and whether the fault affected system safety or functionality

	C	T	CxT
Safety	.4671	.2872	.1342
Functionality	.3224	.6968	.2246
Mean	.3947	.4920	.1794
		Mean of C x Mean of T =	.1942
		Covariance CT =	-.0148

Note: C = mean proportion of faults missed by checkers; T = mean proportion of faults missed by testers.

Notice that in both Table 19 and Table 20 the mean figures for C and T ineffectiveness are the same (and the same as was found from the full set of 16 faults). Therefore, the 'naïve' figure for CT ineffectiveness, i.e. the 'mean of C x mean of T' is .1942 in both tables, because it is simply the product of the average ineffectiveness figures for checking and testing.

The degree to which this 'naïve' figure is an over- or under-estimate of the 'true' figure (.1815) is dependent upon the covariance between the checking and testing processes, i.e. the extent to which they are diverse. However, by grouping the raw fault data in different ways, the measured covariance between checking and testing will differ, as shown below.

As we know from the full data for all 16 faults, the 'true' covariance is -.0127. Based upon the data in Table 19, aggregated by signalling category, the figure is estimated to be -.0010. While this is still in the same direction as the true figure, i.e. negative covariance, it is an order of magnitude too small. Therefore, the best estimate that we can make of the ineffectiveness of an application of CT fault-finding based upon these data is .1932 (.1942 + -.0010), which overestimates the true ineffectiveness of .1815.

The situation is reversed when the data are presented broken-down by safety /functionality (Table 20). Here, the covariance between C and T is -0.0148, and the estimate of CT ineffectiveness is therefore .1794 (.1942 + -.0148). Compared to the true figure of .1815 the covariance was overestimated and so the ineffectiveness underestimated, i.e. the benefits of diversity have been overstated.

Categorising faults in different ways leads to different estimates of the covariance between checking and testing because the categorised data 'package' the variation of fault 'difficulty' *within* fault-class differently. The difficulty of each fault or fault class is shown in Table 19 and Table 20 as the product of the C and T ineffectiveness values, CxT. The CxT index is therefore a measure of the extent to which a fault or fault class is *difficult*. For example, in Table 19 fault class ASP has a CxT value of .2021, and fault class ROU has a value of .1029. Therefore, overall we could consider ASP faults to be nearly twice as difficult as ROU faults.

Taking the mean of all of the CxT values gives an estimate of the overall difficulty of the programme being checked. Notice that it is the same estimate as was arrived at earlier by taking the product of the overall checking and testing ineffectiveness measures and adding the covariance. For example, Table 20 categorises faults by whether they affected the safety or functionality of the SSI system. Taking the product of the mean fault detection performances for checking and testing gives, as always, .3947 x .4920 = .1942. Adding the covariance produces the predicted ineffectiveness of an application of checking and testing: -.0148 + .1942 = .1794. This is the same figure as is obtained by taking the average of the difficulty values (CxT) for safety faults and functionality faults: (.1342 + .2246) / 2 = .1794.

The overestimate of the magnitude of the covariance between fault classes is identical to the average covariance within fault classes. Within safety and functionality classes the mean covariance = [.0128 (safety) + -.0086 (functionality)] / 2 = .0021. Adding the average covariance within classes to the overall ineffectiveness estimate gives .0021 + .1794 = .1815, the true measure of CT ineffectiveness.

This problem of covariance within classes does not arise when using the raw fault data. In effect, each fault is a 'class' all of its own. In this case the covariance of fault difficulty within classes is automatically zero, and therefore none of the covariance between checking and testing is 'hidden' from the attempt to characterise it. Therefore, when using aggregated fault-class data to estimate the true effectiveness of a set of diverse fault-detection procedures, the lower the covariance within fault classes is, the more accurate the estimate of the benefits of diversity will be. As the covariance within classes approaches zero then the estimates produced for diverse performance will approach the exact, true figure.

Further exposition and a formal proof of the details behind this approach to measuring diversity in software are given in Littlewood *et al* (2000). The methods used for characterising diversity, illustrated above, allow proof of the general assertion that, broadly speaking, diversity between methods is beneficial for fault-detection, and more diversity is better than less. The results are also shown to generalise to situations with

any number of fault detection methods, not necessarily just two as was considered here.

How does the above analysis work to benefit software safety in practical terms? Considering just checking and testing for now, each fault-finding process would be applied to a given programme independently and in roughly equal 'amount'. (Quite what is meant here by 'amount' is discussed in Littlewood et al, 2000. For now suffice to say that the methods could be applied until they had detected roughly equal numbers of faults.) The faults detected by each method would then be categorised in some way, and data on the difficulty of each fault category could then be computed. The estimated effectiveness/ineffectiveness of the combined CT process could then be computed. 'Confidence limits' could then be applied to this estimate based upon our beliefs about the covariance between methods within fault class. If we believed that the within-class covariance was zero then the effectiveness prediction would be exact, and as covariance increased then the prediction would deviate accordingly. Data from previous studies and software projects could be used to estimate the extent of intra-class covariance, and ideally classes with close to zero within-class covariance between methods could be selected.

10 General Discussion

This thesis set out to investigate some of the factors that influence the extent to which it is possible to build a reliable human-machine system with unreliable (human) components.

The approach taken was initially context-bound: the system in question was an existing safety-critical software development process. Two phases of research were undertaken: an exploration phase, which sought to describe the system and generate hypotheses, and; an evaluation phase, which sought to test these hypotheses and generalise them. This final chapter of the thesis represents a third phase, where the questions that have arisen due to the work must be addressed.

10.1 Summary of empirical findings

The exploration phase collected ecologically valid but uncontrolled data regarding the SSI DPP. A range of DPP tasks were described, concerned with creating and then assuring the quality of safety-critical computer software. Although the various tasks were found to be error-prone, the system as a whole seemed robust in detecting and eradicating faults from the code. It was suggested that at least part of the reason behind this was the diversity inherent in the fault-finding tasks employed, which may protect the system against the type of common-mode errors observed in the work sample.

Following a conceptual description of diversity a series of controlled experiments investigated possible dimensions that may have been able to contribute to diverse task performance, e.g. spatial and verbal abilities, task instructions, mental models.

The strongest case for the demonstration of cognitive diversity came from the characteristics of the two tasks compared: checking and testing. One of the most consistent findings was that the different fault-finding tasks, which had different representations of the 'problem' (i.e. checking and testing), led to qualitatively different performances in finding faults. A general model of the effects of diversity in fault-finding task performance was presented. This model shows the benefits of diversity as the interplay between intra- and inter-category covariance between fault finding

strategies. It represents an advance over previous models in that the estimation of model parameters should be possible in a practical fashion, based on existing data on fault finding categories.

Among a few other candidate explanations it was suggested that the task representations (checking and testing) might differ in their suitability for spatial vs. verbal processing, and that this might be the source of cognitive diversity. However, some of the empirical findings do not support this view. If it is assumed that, say, checking was more suitable for verbal-linguistic processing, then those high in verbal ability would have been expected to perform relatively better at checking compared to testing. Those higher in spatial ability would be expected to show the opposite association. However, there were no such interactions found between spatial/verbal abilities and task type (checking or testing). This could be because of a restriction in range in participants' verbal and spatial ability scores – all were from educational or occupational populations that are selected on these traits. In addition, the levels of verbal and spatial abilities are not independent, and will tend to be highly correlated.

A more substantial objection to the conclusion that verbal vs. spatial processing was the heart of the observed diversity is found when considering the equivalence of the checking and testing tasks, particularly in terms of their 'size' and 'difficulty'. These topics will be addressed in the remainder of this chapter, where a paradigm for investigating diversity in problem-solving on a surer footing will be outlined.

10.2 Problem size and difficulty

It has been assumed until now that because the checking and testing tasks are different manifestations of the same problem, they must also be comparable in terms of their difficulty (or size, or complexity, etc.). This assumption received support in the laboratory simulation tasks, where neither checkers nor testers were found to outperform the others in overall fault-finding. However, in a seminal paper, Newel and Simon (1972) proposed an analysis of problem-solving tasks that highlighted the role of at least two independent factors that contribute to task difficulty; the size of the problem space and the characteristics of the 'move' operators. It will be argued below

that the SSI tasks cannot be considered to be equivalent in terms of either the size of the problem space or the difficulty of the move operators. Nevertheless, a task paradigm that does have such equivalence will be presented as a format for further investigation of the characteristics of cognitive diversity.

As mentioned above, Newel and Simon (1972) highlighted the problem space and move operators as crucial factors in task difficulty. Problem space is the term given to all possible states of a problem that could be encountered by a problem solver. Move operators are the methods by which the problem solver can transform one problem state into another, i.e. the way in which they can 'move' within the problem space. The problem is then solved by searching through (or constructing an internal representation of) the problem space. Good performance is that which transforms initial to goal state in the smallest number of moves, i.e. with fewest steps in between, or in the shortest time, say.

It was initially thought that the crucial factor affecting performance was the size of the problem space. For instance, chess is considered to be a more difficult game than draughts, and fittingly chess has a problem space which is almost unimaginably large compared to that for draughts. (Chess is estimated to have around 10^{40} different possible games compared to around 10^{20} for draughts; Chellapilla & Fogel, 2001. By way of comparison, the difference between chess and draughts in terms of problem space size is approximately the same (log) ratio as the difference between the possible games of chess and the number of atoms in the observable universe, about 10^{78}.)

Newell and Simon's (1972) theory describes the difficulty of solving the problem as the difficulty in acquiring an effective internal representation of the problem space. Ideally, such a representation should support algorithmic solutions to the problem, where a particular strategy is guaranteed to arrive at the solution. In practice, however, the problem solver cannot represent even a very large portion of the entire problem space internally, and so cannot determine that any particular strategy will definitely lead to the goal state. What the problem solver must instead do is adopt heuristics (e.g. means-end analysis); strategies that move the game into states that seem nearer to the goal state

(but with no guarantee that the goal will be reached). The larger the problem space, it was reasoned, the more extensive the search through the space would have to be, and the more information would have to be stored to build an effective internal representation of the problem space.

Estimates of problem space size can be computed for the checking and testing tasks. The testing task is discrete, meaning it can only adopt a limited number of possible states. Even for the simplified SSI simulator task this number is large. For instance, in the training layout there are 12 track sections, and each one can be either occupied or unoccupied. This gives $2^{12} = 4{,}096$ different arrangements of just the tracks. Including points (4^2 states) and routes (2^4 states) brings the total to 1,048,576 different possible states that the simulator could be in, any of which could potentially reveal an error. This space of over a million different states could be taken to represent the problem space in which the 4 errors are 'hidden'.

Computing the size of the problem space is less easy when considering the checking task. Using the same criteria as for the testing task would not provide a sensible estimate, as we would count only one state (the fixed code). A better measure would be to take the amount of code present as an estimate of the total size of the problem space. Even this presents problems, however, as was found when scoring the performance of checkers throughout the data collection. The code could be measured character by character, for instance. The industry-standard convention, though, is to measure the number of lines of code. The training layout contains 7 lines of code for each of the four routes in the PRR file, and one line per route in the OPT file. The PFM file contains 2 lines for each of the 2 sets of points. There are then 36 lines to inspect for the checkers.

At face value there seems to be a very large difference between the sizes of the checking and testing task's problem states; 36 lines vs. over a million separate states, respectively (for the training layout). The methods for establishing problem space size given above are not very satisfactory, however. For instance, the checkers cannot assume that the code is complete, and so the 'correct' code could be arbitrarily larger

than that they are presented with initially. This may seem a trivial problem in the SSI simulator, where there seems only limited opportunity for extra code (e.g. there are only so many routes, and each route has only a limited number of functions that needed to be coded for). However, quite what would constitute this extra code could be drawn from the entire lexicon of SSI code language.

To illustrate with a natural language example, imagine what might be missing in the following sentence: "Call me _____". It is a very small sentence and so, to use the same metric as was used above for the checking task, we would conclude that it has a very small problem space. Without any knowledge of what the sentence is supposed to be representing, however, there are a very large number of nouns, adjectives, pronouns etc. that would fit into the sentence. Therefore, the mental search space is actually very large. Of course, the SSI checker has some idea what the code is supposed to represent and this reduces the problem space size considerably. However, what remains is still likely to be very large. To illustrate, if I tell you that the sentence above is the first sentence in Herman Melville's 'Moby Dick' (first published under the title 'The Whale' in 1851), does that help? [1]

These issues will be revisited later. For now, it is enough to say that the methods adopted above for measuring problem space size in the SSI task are likely to be more convenient than precise. More damningly, because different methods were used to measure the problem spaces of checking and testing it is difficult to demonstrate their comparability. What is needed is a way to understand the effects of problem-space size on problem-solving in a more rigorous fashion.

Hayes and Simon (1977) provided just such a paradigm. They looked at problem solving behaviour when presented with isomorphic problems – problems with the same problem space size and structure. They found that problem-solving performance differed greatly depending on how the isomorphs were presented to the participants, i.e. what 'cover-story' was given to explain the problem. Similar findings were produced in studies of logic-problem solving. The well-known Wason card-selection

[1] Moby Dick begins "Call me Ishmael."

task (e.g. Wason & Johnson-Laird, 1972) is notoriously difficult to solve optimally, with around 90% of participants failing to do so on their first attempt. However, if the abstract card-selection task, which just involves arbitrary relations between letters of the alphabet, is recast in more familiar terms, then the problem becomes very much easier. Griggs and Cox (1982) found that an isomorph of the card-selection task that involved the relation between age and permission to drink alcohol allowed participants to solve the problem optimally on their first attempt more than 75% of the time. This showed the large effect that the participant's prior knowledge of a problem has on their ability to solve that problem, as long as the problem is represented in a way that allows the problem-solver to recognise the relevance of this knowledge.

So, problem space size alone is not responsible for the ease or difficulty with which problems can be solved. Rather, more recent attention has focused on differences in how move operators are perceived and used by problem-solvers.

Move operators are the methods by which the problem is changed from one state to another, i.e. the methods by which the problem space can be traversed. Kotovsky, Hayes and Simon (1985) showed that it was the memory load imposed by the move making process that accounted for differences in problem solving performance among isomorphic problems with different representations. The higher the memory load the more difficult it was for the participants to "become expert at utilizing the problem rules to make moves..." (p.290). They found that learning (i.e. automation) of move operator information, and thus reduction in working memory load, was necessary before participants were able to start planning ahead and using strategies for solving the problem.

Zhang (1997) extended this work by demonstrating how dissimilar representations of isomorphic problems differ in the amount of move-operator information they provide explicitly and externally (and therefore not imposing working memory load). Difficult representations required the problem-solvers to hold all of the move operator information internally, i.e. in working memory; easier versions represent much of this

information externally, i.e. perceptually, therefore relieving working memory of this requirement.

The previously discussed findings were all conducted using well-defined, or well-formed, problems. Well-formed problems have well-defined problem states, with clearly defined start and goal states, and the operators are also clearly defined. Ill-formed problems do not have such clearly defined parameters; "develop a cure for cancer" would be such a problem.

Kotovsky and Simon (1990) used problem isomorphs that varied in the well-formed nature of their move operators. They used a task called the 'Chinese-ring' puzzle, which involves removing five metal rings from a bar to which they are interlinked with cord. They compared the actual, physical, problem with computer-based 'digital' version that represented the states of the problem explicitly, as balls being moved into and out of boxes (i.e. rings being moved on and off the bar). In one digital version there was no external information about the move operators, in the other the way in which moves could be made was represented explicitly, by showing the lids of the boxes as open when the ball could be moved into or out from the box.

Kotovsky and Simon (1990) found that in the physical ('analogue') version of the task, almost no participants could solve the problem within two hours. Solving the digital versions was completed on average in less than half an hour. The authors found that when attempting the analogue version of the task most participants could not discover what constituted a move, let alone use this information to explore the problem space in search of the goal state. They argued that the three problems were not actually isomorphs until the participants had discovered the move operators. Performances on the two digital versions of the task were found to be very similar once the participants could identify and make legal moves with equivalent efficiency; the version without explicit move information was found to be easier because move-operator information was provided, thus reducing memory load and steepening the learning curve.

Recently, Anderson and Douglass (2002) have also shown that the memory load of different problem-solving strategies affects performance, in addition to the load engendered by the problem representations *per se*. They found that memory for these strategies, involving creating and using sub-goals, is just like any other information in working-memory such as memory for problem states.

Applying work on problem-spaces to the checking and testing tasks we can see that there are marked differences between checking and testing (Table 21).

Table 21. Difference between SSI checking and testing tasks in terms of problem space size, move operator information, and the extent to which both are well- or ill-formed.

	Checking	Testing
Problem space size	Although the code is finite in size, it is not necessarily complete, so in principle the 'correct' version could be arbitrarily large	Finite but large
Move operator information	To make 'moves', the checker must imagine how the code will behave based upon remembered signalling rules. The checker must also be able to envisage how alterations to the code would affect functioning. Memory load is therefore likely to be high.	The move operators are discrete and represented explicitly by the toggle switches on the simulator.
Well formed?	No. The problem space is not defined, and move operators are unlikely to all be available from memory at any one time.	Yes. The problem space is finite and move operators are all defined

So, it cannot be assumed that the checking and testing tasks are isomorphic in terms of either problem space or move operators. Therefore, there is no guarantee that the observed diversity was not because of the action and / or interaction of these factors. To investigate further diversity between spatial and verbal processes, what is needed is a well-formed problem-solving paradigm that has problem representations isomorphic in terms of their problem spaces and the explicit information they provide about move operators, but that are presented in primarily spatial vs. verbal formats.

The remainder of this chapter will present a description and discussion of the "Towers of Hanoi" (TOH) puzzle, and how it could be used to investigate diversity between different representations of problems. This discussion will continue to some length, but will finish having specified particular isomorphic representations of the TOH that will be suitable for studying isomorphic representation of problems, without so many confounding factors as present in the SSI case study.

10.3 Description of the Towers of Hanoi

The TOH was chosen as it is has been well studied by mathematicians and cognitive psychologists, and is well known and understood compared to other, similar problems.

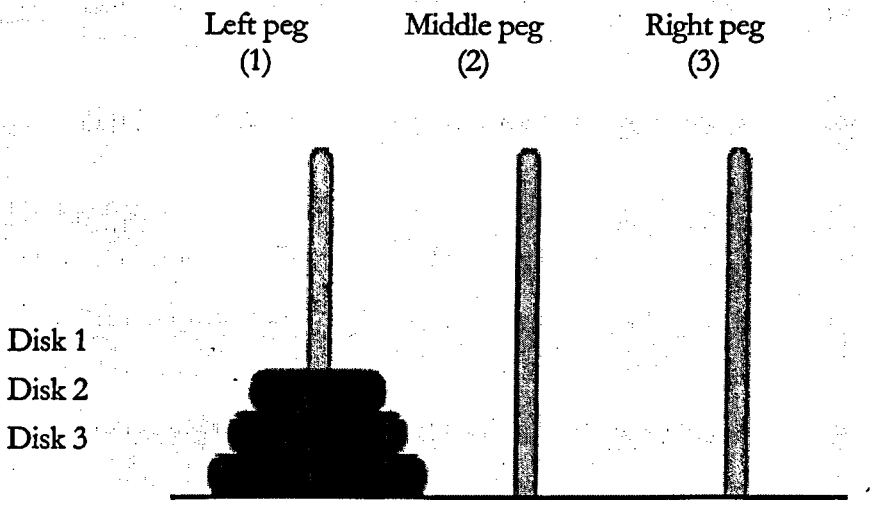


Figure 9: Illustration of the Towers of Hanoi puzzle.

Pegs are labelled Left, Middle & Right; disks are labelled 1, 2 & 3 (3 being the largest).

TOH is played on a board with three vertical pegs, over which fit a number of circular disks of different sizes. Figure 9 shows a three-disk TOH, with all of the disks stacked on the left-hand peg.

The aim of the game is to move the disks from one arrangement to another. This is most often from the starting point where all of the disks are on the left-hand peg, to a finish point where all of the disks are stacked on the right-hand peg. There are a number of constraints on how this can be achieved, however. The disks must be moved only one at a time, and each "move" must transport a disk from one peg to another peg. Also, bigger disks cannot be placed on smaller disks. Finally, only the smallest disk in a pile can be moved on any particular move. These rules can be stated as follows:

> *TOH Rule 1*: One, and only one, disk must be moved on each turn from one peg to another peg.
>
> *TOH Rule 2*: If a peg contains more than one disk, only the uppermost disk can be moved.
>
> *TOH Rule 3*: A disk can only be placed on a peg with no smaller disks below it.

Perfect performance is achieved when the disks are moved from the start to the finish points in the fewest possible moves. In its three-peg, three-disk incarnation, as shown in Figure 9, a sequence of seven moves is the shortest that can be achieved. With more disks and/or more pegs the finish point becomes more 'distant' from the start point and a greater number of moves is required to finish the puzzle. For example, with four disks instead of three, 15 moves are required; with five disks, a minimum of 31 moves is necessary.

The reason why these exact numbers of moves are required to complete games with different numbers of elements (i.e. disk and pegs) can be seen by mapping out the problem space of the TOH in graphical form.

10.4 The TOH problem space

A problem state in TOH is a particular legal arrangement of the disks on the pegs. So, the state of the TOH game shown in Figure 9, with each of the disks on the left-hand peg, represents just one problem state. From this position, moving the smallest disk from the left-hand to the middle peg moves to a new problem state. Because there is a finite number of both disks and pegs (and rules for moving the disks on the pegs) it can be seen that the number of different problem states that are possible in the TOH is also finite – there are only so many different ways of arranging the disks on the pegs. In the case of the three-disk TOH the total number of problem states is 27. This collection of problem states is what was referred to earlier as the problem space. The problem space can be thought of as defining an abstract space in which the different possibilities of the game are bound.

This problem space can be represented by drawing it out in the form of a graph. A graph is a diagram consisting only of points, or vertices, and lines connecting the points, called edges. The TOH state-space will thus consist of 27 vertices, or points, because it has 27 different states that the disks and pegs can be in. The vertices will be joined by edges (lines), each edge showing where one problem state can be reached from another by making a legal move. The most efficient way of drawing out these 27 vertices and associated edges is shown in Figure 10.

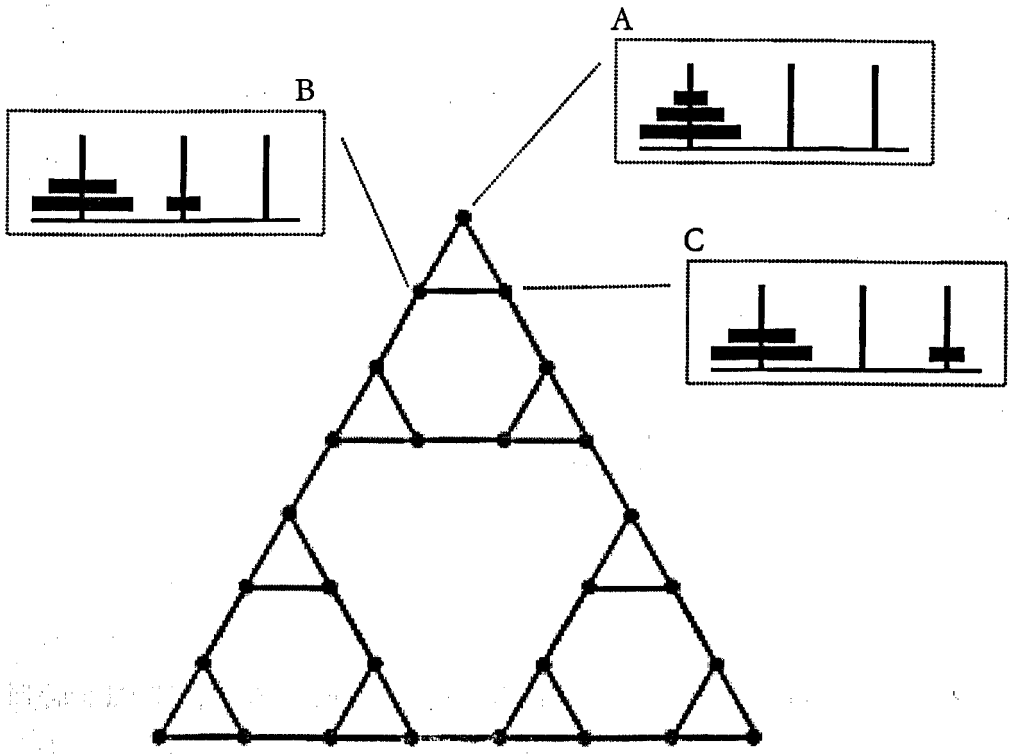


Figure 10: Graph of the Towers of Hanoi problem space

Boxes A, B and C show the arrangement of disks represented by the associated vertices. For example, Box A is associated with the uppermost vertex, signifying that this vertex represents the state of a TOH game where all of the disks are on the left-hand peg. Vertices A, B, and C can all be reached from one another by making a legal move, therefore they are interconnected by edges

10.5 Sierpinski's Gasket

The triangular shape in Figure 10 has been called Sierpinski's Gasket, after a mathematician of the same name who did extensive work on its properties in the early part of the 20th century. (As well as being useful in understanding the TOH, Sierpinski's gasket is fundamentally related to the binomial distribution and the distinctive, serrated shape of the B-2 Stealth Bomber, amongst other things.) The relationship between the Sierpinski gasket and the TOH problem space will be easier to visualise if each of the twenty seven possible states of the game is properly labelled.

However, to do this in the space allowed means adopting a more space-efficient method of labelling than that used in Figure 10.

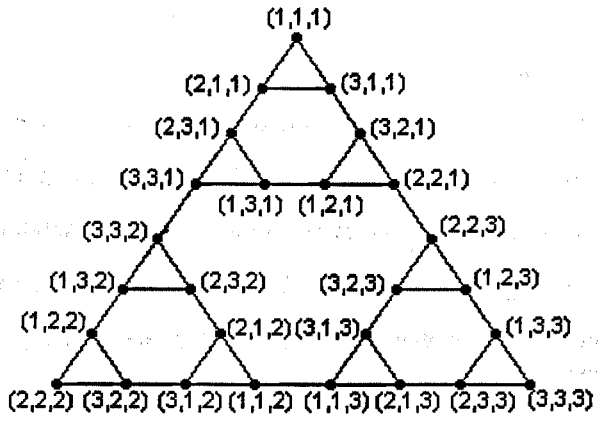


Figure 11: Sierpinski's gasket labelled with TOH problem states.
Each vertex has a three digit label; the first number denotes the peg that the smallest ring is on, 1 being the left-hand peg, 2 being the middle and 3 being the right-hand peg. The second digit denotes the peg position of the next biggest disk, and the final digit the position of the largest disk. The top vertex (1,1,1) represents all the disks on peg 1 (the left-hand peg).

In Figure 11, each vertex has a label denoting the state of the TOH game that it represents. The state with all disks on the left-hand peg, labelled A in Figure 10, is here labelled (1,1,1). The first number denotes the peg that the smallest ring is on (the left-hand peg is 1, the middle peg is 2 and the right-hand peg is 3). The second digit denotes the peg position of the medium-sized disk, and the final digit indicates the position of the largest disk. So, (1,1,1) means that the smallest, medium and largest disks are all on peg 1. The goal state, where all of the disks are stacked on the right-hand peg (peg 3), is labelled (3,3,3) at the bottom right-hand corner.

The edges connecting the vertices represent all of the allowable moves by which the various problem states can be transformed from one to another. So, Figure 11 shows that the starting state, (1,1,1) is connected to (3,1,1) and (2,1,1). This means that the

smallest disk can legally be moved to either the 2nd or 3rd pegs. No other connections are shown; the connections represent the sum total of allowable moves, which in this case are the options available when starting the game with all of the disks on the 1st peg (1,1,1).

A game analogous to the Towers of Hanoi could be played on the Sierpinski gasket. Imagine that a counter, representing the current game state, is placed on the (1,1,1) vertex at the top of a drawing of the gasket such as in Figure 11. The goal is then to move the counter to the (3,3,3) gasket at the lower right-hand side, one vertex at a time, in the smallest number of moves possible. A move is made by moving the counter from the current vertex to an adjacent one, i.e. one connected by an edge to the current vertex.

To complete the game in the fewest moves the player has to able to discern the shortest path, in terms of connected vertices, which links the start point with the goal point. From Figure 11 it can be seen that in this case the game is trivially easy to complete. A straight line of links making up the right-hand edge of the gasket connects the start and finish points in seven moves. Seven moves is exactly the same as needed for TOH because the two games are isomorphic, they have the same-sized problem spaces. Even though the Sierpinski game is isomorphic with the Towers of Hanoi game in terms of problem space size, it is much easier to succeed at because the games differ in how they represent that problem space and their representations of move operators.

10.5.1 Distinctions between the Sierpinski game and TOH

Although the Towers of Hanoi and the Sierpinski games are isomorphic in terms of their problem spaces, the move operator information is very different.

For instance, the Sierpinski game shows the problems states in parallel, i.e. all at the same time, whereas the TOH game shows them sequentially, one at a time. When viewing the Sierpinski gasket we are essentially seeing all the possibilities of the game laid out in a naturalistic, spatial metaphor: a map; where all of the points in the

problem space are presented from the perspective of an observer situated outside of the space itself – a so-called allocentric or 'God's eye view'. This makes the task akin to navigating a 2-dimensional space viewed from a point in 3-dimensional space. A similar scenario is produced when, say, navigating across an essentially flat landscape from an aircraft. All of the points of the problem space are laid out in a 2-D plane 'below' the player. Perceptual processes expert in dealing with such spatially-mapped visual information can then decode the scene into an internal model of the 2-D problem space quickly and without memory load. This model can then be used as input to goal-orientated processing that maps the desires of the player (e.g. to make a 'good' move) onto cognitions and ultimately actions, to reduce the distance between current and goal states. To refer back to Kotovsky, Hayes and Simon's work (1985), when playing the Sierpinski game almost any player with adequate vision will be an expert in terms of the move operators before they have even played the game. In addition, total information about the problem space is available externally and so does not need to be maintained in working memory.

The TOH game, on the other hand, can only represent states discretely; one at a time, through a 'temporal window'. To work out the correct path in the TOH game the passage of sequences of states must be remembered, and future states imagined. When playing the TOH game the 'viewpoint' of the player is situated within the problem space itself, located at the current problem state. This means that the game, while still involving the navigation of a flat landscape (the problem space map, in effect), is instead done from the point of view of a hiker walking across the problem state-plane. This hiker will not have access to the same amount and quality of problem state information as would the 'airborne' Sierpinski game-player.

In both games there is a representation of the distance between different problem states, i.e. the extent to which states of the game are 'near' or far from one another. In the Sierpinski game knowledge about the distance between states is derived from the God's-eye view perspective. Here the states are projected onto a 2-D plane which has all points essentially equidistant from the observer (player). The observer potentially has knowledge of each of the states in equal measure, and as stated above the

information about relations between states (which ones are nearer the goal, say) will be provided by effortless perceptual processes.

In the TOH game, on the other hand, the perspective point of the observer (i.e. where the player sees the game from) is egocentric, situated within the problem space. We can imagine what the TOH 'hiker' navigating across the problem state-landscape would be able to see. They would be able to see their current location, i.e. problem state, directly represented around them. This could perhaps take the form of a place-name. Knowing how place-names work (i.e. knowing the rules of TOH) would give the hiker some clues as to the destinations that could be reached by taking one of the 'paths' leading away from their current location. These paths would represent legal moves, and would lead to different locations on the Sierpinski game-landscape. However, knowledge about the states outside of the current location would depend upon the hiker being able to infer or remember them. It would be fairly easy to find out what adjacent locations looked like, because a path could be selected (even at random) and it would definitely lead to a new location (state) somewhere in the landscape. (Contrast this with the Chinese ring puzzle [Kotovsky and Simon, 1990]. In the analogue version of the game it was not obvious what even constituted a move; to our hiker this is equivalent to not having any paths marked out.) Although knowing the whereabouts of paths, the hiker would have no map, however, and so planning a route would involve having to remember or discover which paths lead to where via which other locations. Distance on this TOH landscape would have a more direct meaning; rather than the observer being located equidistantly from all problem states – as with the Sierpinski game's allocentric view – the hiker is actually nearer to some states and further away from others. Distance here really means the number of locations between the current and goal states, or the time it takes to make all the intervening state-transformations, or some other appropriate sequential (ordinal) relation.

Combining these factors it can be seen that the TOH and Sierpinski games are far from being isomorphic as they stand. The spatial map of problem states in the Sierpinski game represents an external (i.e. not memory dependent) source from which the distance and orientation between the present and goal state can be obtained by

automatic perceptual processes. This then makes the task of selecting the appropriate move no more demanding than finding a route on a fairly simple map.

The TOH game, on the other hand, is sequential and memory dependent. Rather than showing all of the game states at one time, each state can only be inspected while the others are hidden. To obtain strategic information the player must hold internally the relation of their current state with past and future states. For the TOH this means representing the locations and relative positions of the disks on each peg, or at least the differences between their relative positions from state to state (i.e. 'move' information). For novices at the game the amount of information needed to represent the full problem space is likely to overburden conscious short term memory capacity. Only the current problem state will be memory independent, because it is the only one on display at any given time. Knowledge about the other possible game states will thus tend to decrease as their 'distance' from the current state increases. Knowledge about which possible future states are nearer to the goal state than the current state (i.e. planning information) will also vary with the level of knowledge of the local 'territory' of the TOH 'landscape', i.e. problem space .

10.6 A revised version of the Sierpinski game

The differences between the Sierpinski and TOH games, discussed in the previous section, can be removed by representing the Sierpinski game in a different fashion. Instead of the Sierpinski game showing all of the problem space at one time, we can imagine playing it in a dark room with a weak flashlight, so that only a very small proportion of the game map could be illuminated at any one time. If only one vertex label could be seen (and no edges), this situation would be equivalent to the TOH game. In both cases, the only thing that would be visible at any one time would be the current problem state. All other information, such as previous and possible future states, would have to be internally modelled or remembered.

To put the 'restricted view' Sierpinski game described above into a more playable form (without needing dark rooms and flashlights), another game type can be invoked. A suitable choice is the game where words are transformed from one to another by

constructing intermediate words that differ only by one letter from the preceding and succeeding words. For instance, to transform the word DOG into CAT, the sequence shown in Table 22 achieves this in the fewest possible steps (3):

Table 22. A sequence of transformations to turn DOG into CAT.

Move	Description	Word
0 Start	Starting word	DOG
1	Change D to C	COG
2	Change G to T	COT
3 Finish	Change O to A	CAT

Of course, as in the TOH, there are rules governing the moves that can be made. A sequence of three symbols has to be manipulated, one symbol at a time, to produce a goal sequence in the smallest number of 'moves'. Rules governing moves are concerned with which symbols can be manipulated and how: Only one symbol (letter) can be changed at a time; only symbols from the English alphabet are allowed (no "£", "Ж" or "ς", for instance); each intermediate word must be a correctly-spelled English word (no XOG), and; importantly, symbols are substituted for others based upon their position in the sequence, like for like. Therefore, this format can be labelled a "symbolic positional" format. Table 23 shows the "restricted view" Sierpinski game in this symbolic positional format.

In Table 23, instead of the full map of the problem states given by the Sierpinski gasket in Figure 11, there is now just a sequence of adjacent states that represents the shortest possible sequence of moves from starting to goal state. The information from the gasket showing the relationship between the current state and all other possible states,

as well as the paths between them, is now hidden. In this format only the particular arrangements of game elements (here, the number symbols) are visible, just as in the TOH game.

Looking just at the right-hand column in Table 23 (labelled "vertex") the symbolic positional Sierpinski game is represented only as a sequence of symbols arranged into columns. The rules for transforming one problem state into other are based upon the rules outlined earlier in the chapter for the TOH game, but in this new context (i.e. the symbolic positional Sierpinski game) the disks and pegs seem to hold little relevance any more. Surely there must be a set of rules that is mapped onto the purely symbolic and positional world of the vertex labels, without reference to the world of disks and pegs?

Table 23. A winning Sierpinski game sequence in symbolic positional format

Move	Description	Vertex
0 Start	All disks on peg 1	1, 1, 1
1	Disk 1 to peg 3	3, 1, 1
2	Disk 2 to peg 2	3, 2, 1
3	Disk 1 to peg 2	2, 2, 1
4	Disk 3 to peg 3	2, 2, 3
5	Disk 1 to peg 1	1, 2, 3
6	Disk 2 to peg 3	1, 3, 3
7 Finish	Disk 1 to peg 3	3, 3, 3

NOTE: "Vertex" refers to the labels given to denote TOH disk positions, as in Figure 3. Each vertex has a three digit label; the first number denotes the peg that the smallest ring is on, 1 being the left-hand peg, 2 being the middle and 3 being the right-hand peg. The second digit denotes the peg position of the medium-sized disk, and the final digit the position of the largest disk. The first vertex (1,1,1) therefore represents the TOH state where all of the disks are on peg 1 (the left-hand peg).

10.6.1 Abstract rules for the symbolic positional Sierpinski game

Inspection of the TOH rules should reveal the basic functions that they perform and the operators on which they act. A set of rules describing the conduct of the familiar TOH game rules is shown below.

10.6.1.1 Rules for TOH

Synopsis: The game is played with three disks (small, medium and large) arranged on three pegs (left, middle, right). Any peg can contain any disk. The goal of the game is to move a starting stack of disks onto another peg (the goal state) using only the rules below for moving disks.

> *TOH rule 1*: One, and only one, disk must be moved to another peg each turn. (There are three pegs: left, middle and right.)
>
> *TOH rule 2*. If there is more than one disk on a peg, only the topmost disk can be moved.
>
> *TOH rule 3*. A disk can be moved to any peg not containing a smaller disk.

Each rule is concerned in essence only with the relationship between disks and pegs. Rule 1 deals with the basic fact of translation of disks among pegs; rule 2 is concerned with the preference of disks within pegs, and; rule 3 deals with the preference of disks among pegs.

By mapping the above relations onto the symbolic and positional elements of the Sierpinski game, it can be seen that the symbols used in the Sierpinski game (1, 2, and 3) encode the peg 'information' in the TOH. The positions of the symbols in the vertex label (first character, second character, third character) represent the identities of the disks in the TOH (column 1 = small disk, column 2 = medium disk, column 3 = large disk).

By substitution, the rules for the symbolic positional Sierpinski game are as follows:

10.6.1.2 Rules for symbolic positional Sierpinski game

Synopsis: The game is played with three symbols (1, 2 and3) arranged in three columns (left, middle, right). Any column can contain any symbol. The goal of the game is to

transform a starting set of symbols into a goal set using only the rules below for changing symbols.

> *Sierpinski Rule 1*: One, and only one, symbol must be changed each turn into another symbol. (There are three symbols: 1, 2 or 3.)
>
> *Sierpinski Rule 2*: If more than one column contains the same symbol, only the leftmost of those can be changed.
>
> *Sierpinski Rule 3*: A symbol can be changed into any symbol that does NOT appear to its left.

The similarity between the TOH and Sierpinski game rules is evident, even if some of the operators are differently labelled (e.g. 'peg' instead of 'symbol'). For instance, the information contained in TOH rule three just states that a bigger disk can't be placed on a smaller one.[2] In terms of the Sierpinski rules, rule three states that a symbol can only change into one that's not found on its left. "Left" in the Sierpinski game is equivalent to "smaller" in the TOH. The concepts of "disk" or "peg" are no longer necessary to play the Sierpinski game. Even the term "move", meaning changing the game from one state to another, has its genesis in actually moving the game pieces, which no longer is taking place. Instead, the player is "changing" or "switching" symbols, or whatever appropriate term is preferred (cf. Hayes and Simon, 1977; some isomorphs of the TOH were 'move' problems involving moving game elements, other isomorphs were 'change' games).

So, we now have a version of the symbolic positional Sierpinski game in which the representation and rules are fully isomorphic with the TOH game. Well, not quite.

There is one major distinction between the TOH game and the symbolic positional Sierpinski game as described above. Above, the history of previously used / visited symbols is visible. This is, in effect, a view of the 'path' from the initial state to the current state. This is information equivalent to our Sierpinski hiker keeping a diary of place-names visited, and doesn't directly provide information about the path to the

[2] The actual TOH rule 3 is stated as a negation: "A disk can be moved to any peg not containing a smaller disk", but only to efficiently accommodate the situation where there are no disks on the destination peg. If the rule were stated only as relation between two disks (e.g. "you can't put a larger disk on a smaller one"), the relationship

goal state. It could be considered irrelevant to problem resolution, therefore. However, there is no such equivalent in the TOH game, and so the history information should be removed to preserve equivalence between the symbolic positional Sierpinski game and the TOH. In practice this would mean being able to view only the last row of symbols at any one time in the symbolic positional Sierpinski game.

With this fully isomorphic version of the TOH, the only differences that remain between the Symbolic Positional Sierpinski (SPS) and TOH games are in terms of the stimuli used to represent the games states, for both input modality and output modality, and the associated learning that these stimuli invoke.

10.6.2 Stimuli in the TOH and SPS

The stimuli in TOH games were traditionally physical in nature, i.e. actual pegs and disks. These days research into the TOH is often conducted with computer based visual / spatial simulations of the 'physical' TOH (primarily because it allows greater ease of data collection). The required behaviours in the physical game involve using, say, a hand, for direct, physical manipulation of the game pieces. In the simulated TOH the required behaviours are very similar, but in most cases involve indirect manipulation of the pieces using a mouse or similar pointing-device. (Manipulation of the game is indirect in the sense that there is an additional layer of stimulus-response mapping required – hand to mouse to visual feedback, vs. hand to visual feedback in the physical TOH game.)

The visual stimuli available to the player of a simulated TOH game can be made arbitrarily similar to the stimuli in the physical game, but usually the representation is fairly low fidelity. Analogous schematic representations, such as that used in Figure 9, usually suffice. In terms of transitions between game states, a low fidelity simulation could omit these and merely show the various states as snapshots that could be cycled-through akin to a slide-show (but here with access to the states being governed by the rules of the game). Higher fidelity simulations could be used, perhaps to represent the

would be undefined when there were no disks on the destination peg. Alas this is the way that null hypothesis significance testing works, too

transitions between problem states by, say, allowing mouse-mediated 'direct' manipulation of the pieces on-screen, i.e. using the mouse to 'move' the disks.

It is easy to imagine how the SPS game could be similarly implemented. A computer would show the current state of the game on a screen, consisting of the three symbols. The state that the player would like to move to could be input via the keyboard (if numeric symbols were used, as they are in Table 2). Alternatively, the options for selecting or changing symbols could be made part of a graphical user interface that could be manipulated with the mouse, akin to the TOH simulations discussed in the previous paragraph. When the player had selected a move the screen could merely change to the new representation (if a legal move was specified), which would make the SPS game equivalent to the low-fidelity TOH simulation.

In low-fidelity simulations, where transitions between states is not represented, for both the TOH and SPS cases there would be 27 different screens that could be displayed, i.e. the 27 states of the problem space . Each screen would show the representation of that state in the format appropriate to the game (SPS or TOH). Movement between states (screens) could be controlled in both cases by identical behaviours using a mouse controller. In this case the two games, TOH and SPS would be fully isomorphic, identical in all but the stimuli used to represent the different problem states.

10.6.3 Representation of problem states

It is argued informally above that low-fidelity simulations of both the TOH and SPS games, as described, represent isomorphic problems that differ only in the particular representations of their problem states (and therefore also the different associations that these will bring to mind).

The visual stimuli used on the computer-based TOH are designed to evoke a spatial metaphor based upon the physical presence and properties of real disks and pegs (e.g. movement: pegs are fixed; disks can be moved). The metaphor is useful because it seamlessly encodes much of the move operator information that is usually taken for

granted. For instance, we fully expect that the disks won't start moving around on their own, because our expectations of disks and pegs probably precludes movements of any kind without the actions of some external force (e.g. us, the wind, an earthquake). More fundamentally, we probably don't expect the disks to be able to be placed into the same physical space on the peg; only on-top of or underneath one another, for example.

The TOH simulation physical metaphor also provides useful information about the actual game rules. For instance, as well as encoding disk identity, the physical size of the disks is used by the third TOH rule (no bigger disks on smaller ones), and so the very appearance of the disks provides a memorial hook for the rule. This information is always available externally, too, because there are always three different sized disks to look at, whatever the game state. It still must be remembered by the player that larger disks don't go on smaller ones instead of the converse (i.e. smaller disks don't go on larger ones), but stacking things into piles with the biggest things at the bottom, not the top, is probably a well-learned association before the TOH game is ever encountered. There are versions of the TOH where this rule is even more deeply embedded in the game metaphor; instead of disks there are cups of different sizes (or Russian dolls) that must be fitted inside one another, and so the converse to the rule cannot physically be made to occur.

A well as encoding useful information, the physical metaphor of the TOH is also responsible for mistakes that commonly occur when first playing the game. For instance, distance between the physical elements of the game does not correspond to distance in the problem space. Therefore, moving a disk physically nearer to the goal peg does not always move the game nearer to conclusion (this is how a strategy based upon a means-end heuristic could fail, for instance). In contrast, in the game played with a counter on the Sierpinski gasket map, moving the counter nearer to the goal always shortens the path to the goal (if played from the 1,1,1 starting position).

The SPS game does not have such reliance on a physical / spatial metaphor. Disk information *is* represented spatially, or at least sequentially, as disk size increases from

left to right in the vertex label. However, peg information is encoded by the identity of the visual symbols. It is identical in this respect to written alphabetic language, where both the identity and position information of a symbol within a word are necessary to identify the word. (For example, the words "dog" and "god" are not synonymous, despite being composed of identical symbols.) Using the same analogy, words in an alphabetic language are equivalent to problem state labels (vertex labels) in the SPS game. For this reason it is quite likely that some or all of the very well learned skills that we acquire for dealing with words, or numbers, will be used when playing the SPS game. Metaphors based upon linguistic or mathematical knowledge are likely to dominate.

This linguistic metaphor will also provide some of the taken-for-granted move operator information. As is found with alphabetic writing, the SPS labels are symbolic and positional. Therefore, general information, such as the fact that symbols can be substituted for others and that they have different meanings in different positions, can be inferred by a player that can recognise the problem format (i.e. a player that can read; notice, however, that it will be the problem format, not the problem space, which must be recognised.)

In contrast to general game information, the specific game information, i.e. the specific rules for manipulating symbols, is less easy to map on to a widely known metaphor for the SPS game. Such a metaphor would allow us to effortlessly 'see' why the sequence of SPS labels (1,1,1) (3,1,1) (3,2,1) (2,2,1) is better than, say (1,1,1) (3,1,1) (3,2,1) (1,2,1), in the same way that the Sierpinski gasket allows us to see the straight vs. the crooked path to the goal.

In the same way as for the TOH spatial metaphor, there are some properties that we can attribute to the SPS symbols that could provide heuristic information. The spatial metaphor can lead to the erroneous expectation in TOH that moving a disk nearer to the goal peg is always equivalent to moving nearer to the goal state. If the symbols used in the SPS vertex labels are given numerical status (i.e. at least ordinal, as opposed to

categorical scale properties) then changing a symbol for a 'higher' one ('nearer' to the 'goal' symbol, 3) would manifest itself as a similarly flawed heuristic in the SPS game.

Given these differences, the TOH and SPS games could be expected to provide a useful paradigm for comparing spatially-based vs. verbal/numerically-based problem solving. The reasoning outlined above suggests that these tasks, if implemented as described, would be identical in virtually all respects except the processing 'codes' used, symbolic/spatial (SPS) vs. spatial (TOH). Performances on the TOH and SPS games would then be expected to vary, for instance, with the spatial and verbal abilities of the players.

To put the TOH and SPS games into a form more relevant to the programme-debugging tasks of the SSI DPP, they could instead be set as fault-finding tasks. The rules of the games could be cast as the rules of operation of the game simulation. Faults could be seeded into the simulations of the games that allowed, say, illegal moves between certain problem states. The relative efficacies of the two games in fostering detection of errors of various types could then be judged.

If the spatial (TOH) and verbal (SPS) versions of the game support processing metaphors that detect non-overlapping sets of errors, then cognitive diversity will have been demonstrated. If, after accounting for speed-accuracy trade-offs, the error sets do not show diversity then it can be argued that the representations are functionally equivalent. In such a case it would be interesting to speculate on whether an intrinsic feature of the information processing system had been encountered, one based upon the information as opposed to the modality that delivers (and pre-processes) that information. Just such speculation will be indulged in the next section.

10.6.4 Information structure of the Sierpinski gasket

What, then, might be the fundamental properties of the 'information' contained within the TOH and SPS games? A reasonable starting point to answer this question is to look at the elements of the games themselves. Each game consists of two sets of three elements: One series encodes disk/column information (three disks, three columns);

the other encodes peg/symbol information (three pegs, three symbols). However the sum of problem states that could be encoded with such a scheme is only 3^2, or nine states. This is not enough to code the 27 possible states of the games.

A clue to the location of the extra information is contained in the rules, of which there are three. As well as specifying relations *between* disks (3) and pegs (3), they also specify the rules for disks *within* pegs. There are also three ways in which disks can be arranged on a peg. The position of the largest disk is entirely redundant – if on a peg it always occupies the bottom position – and so it carries no informational 'load'. We need one piece of information to decide where the medium disk is located; if it's not on the bottom itself it must be one up from the bottom, on the largest disk. We need to ask two questions (two bits of information) to find out where the smallest disk is, however. First, is it on the bottom? Second, is it one up from the bottom? If the answers are both 'no', then it's on the top of the pile. A 'yes' at any point obviously locates the disk. So, an additional dimension of information is required to code these three possibilities. This gives a total of three sets comprising three states each that have to be encoded; $3^3 = 27$, the number of problem-states in the TOH and SPS games.

In the TOH game the problem-state identity information is carried by the 27 possible, legal, arrangements of the game pieces. In the SPS game the information is carried by the 27 possible arrangements of the three symbols in their three-column labels. Either game can then be said to have embedded the problem-states into a three-dimensional space, each dimension having three points. (The use of the term 'three-dimensional' is not meant here as the familiar, Cartesian, spatial, physical world which we inhabit, but rather an abstract problem-space with three independent parameters.) We could map the problem space into differently dimensional spaces, however.

Instead of labelling the Sierpinski gasket as we have done up to now, we could instead simply label the vertices sequentially from top to bottom and left to right with the letters of the English alphabet plus another symbol, the Greek symbol omega, 'Ω' (say), giving the required 27 symbols. This would translate (1,1,1) into [A], (2,1,1) into [B], (3,1,1) into [C] and so-on until (2,3,3) into [Z] and finally (3,3,3) into [Ω]. The SPS

game with such coding would mean that each game state would be represented by a single symbol. Instead of a symbolic positional coding scheme, a purely symbolic coding based on one dimension with 27 states (symbols) would remain. This would embed the TOH problem space in an abstract one-dimensional space with 27^1 points, i.e. 27. (Having two symbols instead of one would give a possible 27^2, or 729 states for a hypothetical TOH-27^2 game) The game player would then be tasked with changing A to Ω in the fewest legal steps. (Defining what constitutes a 'legal' step, i.e. the rules of the game, I'll leave to the interested reader. The winning sequence would be: [A](start), [C], [E], [I], [K], [O], [S] [Ω](finish).)

So, if the problem space can be mapped onto any number of abstract informational dimensions, is it important which particular scheme is adopted when considering how easy it would be for humans to navigate the space? The answer is perhaps to be found in the manner in which the problem is (quite literally) addressed.

10.6.5 Hierarchical addressing of the TOH problem space

So far, the addressing systems used to label the TOH/SPS problem states have all had their semantic roots in the physical manifestation of the TOH game (i.e. disks and pegs). This allows the isomorphic nature of the problem spaces to be clearly seen. However, as was demonstrated with the alphabetical labelling, this is not the only way. Any scheme that had enough information-carrying ability (i.e. to represent 27 separate states) would be equally valid.

A brief inspection of Figure 10 shows that the entire Sierpinski gasket is shaped like a triangle. However, the relation to a triangle does not stop there. The big triangle is composed of three smaller triangles, and each of these is composed of triangular triplets of vertices. The Sierpinski gasket is self-similar – small bits of it are identical in structure to larger bits. If we were to add another disk to the three-disk TOH problem, we wouldn't need to redraw the problem-space map (i.e. Sierpinski's gasket), we could just take Figure 10 and photocopy it to produce three copies of equal size, which could be arranged into bigger triangle with 81 (27 x 3) vertices – the problem space of the four-disk TOH.

This recursive self-similarity could be used to label the vertices of Figure 10 with an addressing system that would make this structure obvious. Again, three symbols in positional format could be used to label the vertices, say A, B and C. However, the symbols would not represent disk on peg position, but rather the address of the vertex within the problem space. So, reading the symbols in conventional fashion from left to right, the first symbol would show the location of the vertex in the smallest triangle. The second symbol would show which of the three medium sized triangles the vertex was in, and the final symbol would show in which of the three largest triangles the vertex was located.

To borrow Luger's (1976) terminology, each triplet of three vertices represents the problem-space of a one-disk TOH, i.e. the one-disk subspace. This means that it represents the three choices that we have when 'arranging' one disk on the three pegs. Similarly, each group of three triplets represents a two-disk subspace, and the entire problem space is a three-disk subspace, because it represents the three-disk TOH.

To put this into practice, within triangles of the same sub-space 'scale', the top-most corner could be labelled "A". Moving clockwise, the bottom-right corner would be labelled "B", and the bottom-left, "C". Figure 12 shows how such a hierarchical addressing system would appear.

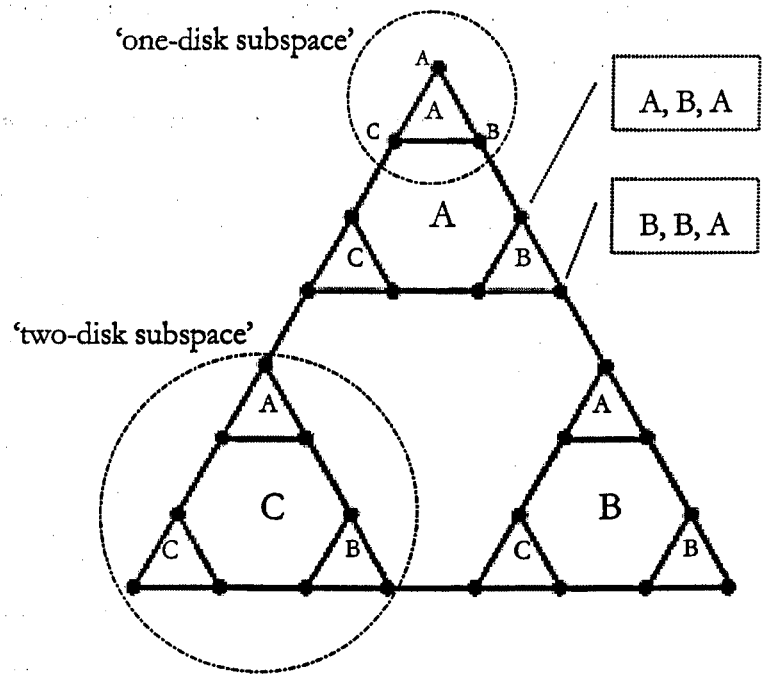


Figure 12. The three-disk TOH problem space labelled hierarchically

The problem space should be conceived as consisting of three big, linked triangles, each made up of three smaller linked triangles (one set of these is labelled 'two-disk subspace'), each of which is made up of three linked vertices ('one-disk subspace'). In each 'subspace' the uppermost corner is labelled A, and then on in alphabetical order clockwise; B for the bottom-right corner, C for the bottom left corner. For clarity only the vertices in the top triplet are shown labelled with 'one-disk subspace' labels. The full, three symbol 'address' is shown for two of the vertices (boxes).

In effect Figure 12, uses the same labelling system that is used for postal addresses. For a postal address, we give a separate piece of information corresponding to the location of our target address, on a number of different 'scales'. Later lines of the address refer to larger-scale constructs, earlier lines refer to smaller-scale constructs. For instance, the first line of a postal address usually locates a building in a street. The next line locates, perhaps, the street within a town, and the next a town within a country (and so on, to an old school-favourite conclusion: "...Earth, solar system, Milky Way...").

Similarly, the first symbol in a hierarchical TOH vertex address locates the vertex within a one-disk subspace; the next symbol locates the triplet within the two-disk subspace, and; the final symbol locates the vertex within the entire, three-disk problem space. So, the vertex (B, B, A) is the bottom-right vertex in the bottom-right corner of the two-disk subspace, located in the uppermost corner of the full three-disk problem space. The shortest winning path using these labels is shown in Table 24.

Table 24. A winning TOH game using hierarchical labelling

Move		Vertex
0	Start	A, A, A
1		B, A, A
2		A, B, A
3		B, B, A
4		A, A, B
5		B, A, B
6		A B, B
7	Finish	B, B, B

NOTE: "Vertex" refers to the symbolic positional hierarchical labels given to denote TOH disk positions. Each vertex has a three-letter label; the first letter denotes location of the vertex in the smallest problem space division (A for the top, and B and C for bottom-right and bottom-left, respectively). The second letter refers to the location of the vertex in the next largest region of the problem space, and the final one the location of the vertex in the largest division. The first vertex (A, A, A) therefore represents the TOH state where all of the disks are on peg 1 (the 'start' position). The final vertex (B, B, B) denotes the goal state, located at the bottom right-hand corner of the problem space.

It is immediately obvious that at no point does the winning sequence require that the C symbol is ever used. This gives a key insight into the dimensional structure of the information represented. Previously, the TOH task was likened to a hiker navigating a

2-dimensional plane (the problem space). However, what the previous TOH labelling schemes didn't make obvious is that the shortest-sequence of moves is in fact just a straight line drawn through this space. A plane has two dimensions, but a line has only one. By using a labelling scheme that represents the self similarity inherent in the problem space, a whole dimension of the space is now redundant.

So, the hierarchical labelling of the Sierpinski gasket seems to have revealed the simple structure of the problem space. Before going on to show how all of this gaming and labelling can be used to understand human problem solving, it is worth showing just a couple more hierarchical labelling systems for the Sierpinski gasket. The first is shown in Figure 13.

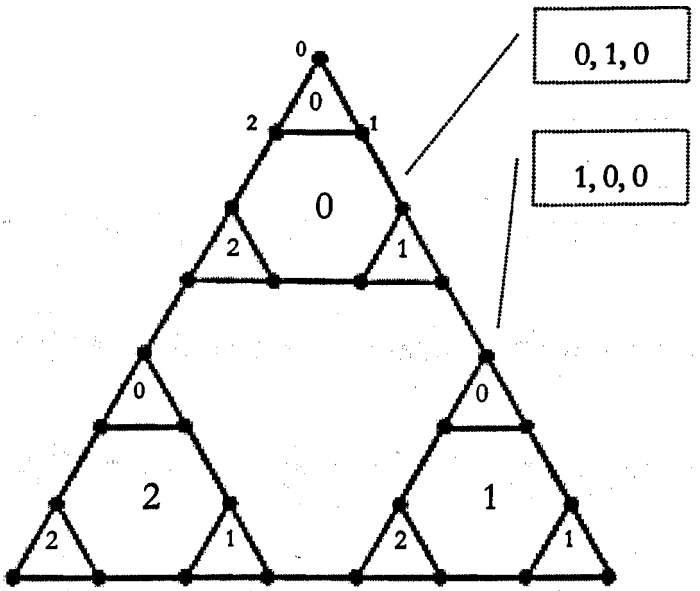


Figure 13. Hierarchical numerical labelling of the Sierpinski gasket. As in Figure 12, the space should be conceived as consisting of three linked triangles, each made up of three smaller linked triangles, each made up of three linked vertices. In each triangle the uppermost corner is labelled 0, and then on in numerical order clockwise; 1 for the bottom-right corner, 2 for the bottom left corner. For clarity only the vertices in the top triplet are shown labelled with 'one-disk subspace' labels. The full, three symbol 'address' is shown for two of the vertices (boxes).

In Figure 13 only two changes are made to the scheme outlined in Figure 12. Instead of the letters A, B and C, the figure is labelled with the numbers 0, 1 and 2, respectively. Then instead of the first symbol from the left representing the smallest 'scale' (triangle), the first digit on the right will show this information. The left-most digit then gives information about the vertex location in the largest triangle (the largest scale). This just means that where previously the labels were read from left to right (meaning small-scale to large-scale information), the labels are now read from right to left to give the same meaning. The shortest sequence of moves to transform 0, 0, 0 into 1, 1, 1 is shown in Table 25.

Table 25. A winning SPS game in numerical hierarchical format

Move		Vertex
0	Start	0, 0, 0
1		0, 0, 1
2		0, 1, 0
3		0, 1, 1
4		1, 0, 0
5		1, 0, 1
6		1, 1, 0
7	Finish	1, 1, 1

NOTE: "Vertex" refers to the symbolic positional hierarchical labels given to denote TOH disk positions. Each vertex has a three-letter label; the right-hand digit denotes location of the vertex in the smallest problem space division (A for the top, and B and C for bottom-right and bottom-left, respectively). The middle digit refers to the location of the vertex in the next largest region of the problem space, and the left-hand digit the location of the vertex in the largest division. The first vertex (0, 0, 0) therefore represents the TOH state where all of the disks are on peg 1 (the left-hand peg). The final vertex (1, 1, 1) denotes the goal state, located at the bottom right-hand corner of the problem space.

In contrast to the labelling schemes used before, it is now quite possible that you can actually 'see' which the good moves are (depending on your familiarity with binary numbers). This is because the winning sequence is now mapped perfectly onto the task of counting in base 2. The labels map perfectly because in effect they specify the same thing. On the one hand the labelling is designed to show the coordinates of the vertices in a 2-dimensional space. However, the winning sequence can be represented as a 1 dimensional line; the line across the full problem space from start point to goal point. This can in effect be seen as the distance from the start point (0, 0, 0, or just 000) to the goal point (111) in terms of how many moves along it is. To decode this information in more familiar decimal format, we just convert the label from binary to decimal as though it were a number. Therefore, the start point, 000, is zero distance from the start. The next move, 001, is one move away from the start. The label 010 is binary for 2, and the similarly-labelled vertex is indeed two moves from the start. The goal state, 111, is just seven in decimal notation. So, this number just represents distance in a one dimensional space, where you can go just forward or backward, towards the goal or away from it.

On the other hand, if the labels can be seen to represent distance from the goal state, they can also be seen as purely specifying some quantity (not necessarily 'distance' in any Euclidian sense) that is an index of how complete the game is. If this quantity is represented as a natural number (albeit in binary notation), then we can count upwards (towards the goal) or downwards (away from the goal). Again, only one dimension of information is invoked; the information given by the vertex label regarding the quantity of 'game completeness'.

In the binary counting version of the SPS game we can use our pre-learned knowledge of the familiar symbolic, positional number system that we use to represent everyday numbers. This will give us implicit knowledge of how we can manipulate the symbols to achieve the operators of 'add one' or 'subtract one', in the same way that implicit knowledge of physical objects gives relevant information in the physical TOH game.

The above is actually a simplification of any plausible use of a counting paradigm to encode the rules for playing the hierarchical SPS game. For completeness, we must consider that the actual problem space is not just the 1-dimensional shortest-path line (represented in binary). Therefore, move operators must allow movement across the 2-dimensional space (which would require the use of all three symbols; base 3). For example, any move from one vertex to the adjacent one to its bottom-right would be a move towards the (1, 1, 1) goal, and would in effect be a binary addition. Similarly, any move that took the player *off* the winning path by moving to the bottom-left (e.g. 011 - 012) would again be the addition of 1, but this time in base 3, as the extra dimension of the problem space is called into play. Moving down the path from the top (0, 0, 0) to the bottom-left vertex (2, 2, 2) is a one-dimensional path just like the winning path. However, it would involve more complex addition in base 3 than just adding one to account for the following sequence of four moves that starts the path: (000) (002) (020) (022). This seems to consist of adding two, then four, then two again. (Actually, this is just binary addition using the symbols 0 and 2 instead of the more conventional 0 and 1). 'Horizontal' moves between states (e.g. 012 - 021) are even more difficult to accommodate within a simple addition-based cover story for the game rules.

In the above case of the numerical hierarchical labelling, as in the other cases discussed, the metaphor brought to mind (counting) is not mapped perfectly onto the problem space. As well as the difficulties raised above, the counting paradigm might lead a player to believe that, if their job is to change 000 into 111, and they are allowed to add one or two or four (in base 2 or 3) each go, then they might get to the goal in only three moves by adding four then two then one (equalling seven [binary]; the goal state). This is not the case.

In effect, by telling participants to transform 000 to 111 by adding one in binary each 'move' is to give explicit instructions to the problem solver as to the correct direction to take across the problem space, and the operators to use to get there, to solve the problem.

A set of operators can quite easily be constructed however, that would allow the effective manipulation of the abstract symbols that are used to give address-labels to the vertices in game-legal ways. A set of rules that works generally is described below.

10.6.5.1 Abstract rules for a hierarchically labelled SPS game

Synopsis: The game is played with three symbols (0, 1 and2) arranged in three columns (left, middle, right). Any column can contain any symbol. The goal of the game is to transform a starting set of symbols (0, 0, 0) into a goal set (1, 1, 1) using only the rules below for changing symbols.

> *Hierarchical Rule 1*: The right-most symbol can be exchanged for any other symbol.
>
> *Hierarchical Rule 2*: The middle symbol and the symbol to its right can be exchanged.
>
> *Hierarchical Rule 3*: The left-hand symbol can be exchanged (identity for identity) with the two symbols to its right, if they are both the same.

These then, are general rules for manipulating arbitrary symbols that represent information in a recursively-addressed problem space. However, the symbols used so far (letters and numbers) will cloud our ability to make inferences about the effects of the informational structure on task difficulty in any experiments that are conducted using these tasks. This is because of the pernicious effects of prior learning – any experience that participants may have with alphabetic or numerical symbol systems is likely to drive their internal problem representation, and thus bring with it knowledge, assumptions, and skill in using move operators that will be hard to control-for and may not be valid for the problem context. Rather than try and seek out adult participants who have had no experience with any symbolic positional information system (i.e. no written language or number system!) it would probably be easier to use symbols or analogies that do not carry such 'baggage'. The use of coloured patches as stimuli has a long and distinguished history (e.g. Stroop, 1935), and would seem a good choice in this case. Instead of the symbols A, B C, or 1, 2, 3, patches of three different colours could be used.

Using colour patches would allow comparison of reconcilable symbolic positional systems. It would still fail to capture the spatial move operator information inherent in the original TOH game, though. This would make inference difficult when trying to separate the effects of the new perceptual formats from the effects of the informational manipulations given by the hierarchical labelling. However, as always, a new representation can be derived; Figure 14 shows such as a system. It is based on the hierarchical labelling of the Sierpinski gasket, but represents move operators as mechanical, spatial actions, much like in the original TOH.

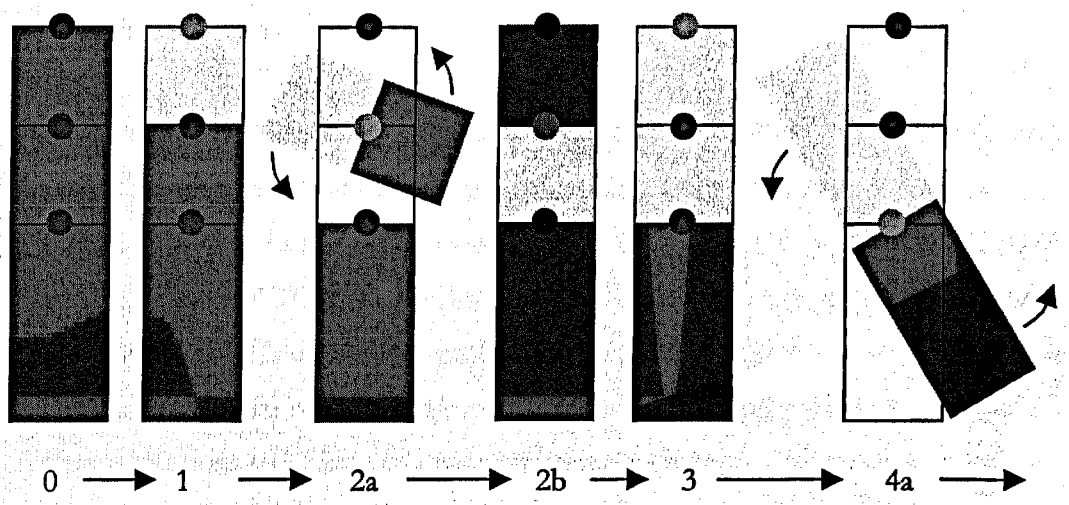


Figure 14. The first four states in the shortest sequence to transform the column from 'red' to 'blue' (light grey), using the hierarchical TOH rules. The three hierarchical rules are represented by the black dots (which are grey when in operation). Step one uses the first rule to transform the upper block to the goal state (blue [light grey]). Step two uses the middle rule to swap the upper and middle blocks (two pictures, 2a and 2b are shown, to show the operation of the rule). Step three uses the first rule again. Finally, step four shows the initiation of the use of rule 3, which swaps the upper two blocks for the lower one. Numbers with arrows show the step, numbers in brackets show the hierarchical numerical label from Table 25.

The representation shown in Figure 14 merely implements the hierarchical rules shown in this section, and is so is isomorphic with the original TOH problem. Each problem state is represented by a column divided into three blocks. The colour of the block (in this case red [dark grey], green [not shown] and blue [light grey]) denotes identity of the symbols used to label the problem space. The top block shows the address of the problem state within the one-disk subspace (i.e. the right-hand symbol of the hierarchical label). The middle block shows the location within the 2-disk subspace (middle symbol), and the bottom block shows the location of the state in the three-disk subspace. Like in the TOH, the rules are represented physically in the game; the rules consist of axes, around which the rule operates to rotate the symbols to their appropriate locations.

As shown, Figure 14 actually has a combination in the way that rules are represented. Rule 1 'changes' the colour of the top block; rules 2 and 3 'move' the blocks to different locations. As such it is an admixture of Kotovsky, Hayes and Simon's (1985) 'move' and 'change' TOH isomorphs. Kotovsky *et al* found change problems to be roughly twice as difficult as move problems. They interpreted this as evidence for the preference for spatial coding of information, which allows information encoded in different locations to be kept relatively interference-free from information associated with other spatial locations. Change problems were thought to require the storage of volatile (i.e. subject to change) information at each spatial location, which would be more prone to decay and interference. It is possible, however, that the participants were just much more practiced at encoding the movement of objects in our spatial world than we are at encoding any wholesale transformations that they may undergo (depending on our expectations for that object; I don't expect wooden blocks [or crystal globes] to be able to mutate in size, but I do expect a balloon to be able to, for instance).

Altering the isomorph in Figure 14 so that rule one operated by actually 'swapping' the top block for one of a different colour would implement an entirely 'move'-based problem representation. By not animating the block transitions, and instead just showing them changing colour, would implement a completely 'change'-based

isomorph. This would allow a more abstract and perhaps more robust test of the 'move' vs. 'change' issue highlighted by Kotovsky, Hayes and Simon (1985).

Returning finally to the justification for this detour into problem solving lore, the problem representation shown in Figure 14 should be directly comparable with the representation shown in Table 24 and Table 25. All have identical problem spaces and move operators, the only differences to be found are in the fundamental skills that encode much of the procedural knowledge for using the move operators. In the example in Figure 14 these are the skills involved in dealing with spatial relations of solid objects. In Table 24 the skills for dealing with discrete symbol systems are invoked; in this case an alphabetic one. The format shown in Table 25 again rests upon skill in dealing with a discrete symbol system, but this time one designed to represent magnitude, i.e. number.

Embedding these representations in a computer programme would allow 'bugs' to be introduced that would affect the outcome of the operations of move operators. These 'bugs' would then be the target for fault-finding efforts which could be compared across tasks. The interdependence between fundamental spatial and verbal / numerical skills in aiding fault-finding and diversity in fault-finding could then be ascertained.

Additionally, the representations discussed above, all using hierarchical addressing of the problem space, could be compared with 'traditionally' addressed representations that separate global vs. local knowledge, e.g. as described in §10.6.1.2. It is likely that the difference in structure of the apparent problem space would lead to different mental models of the problem, which could then facilitate the use of diverse strategies for task performance.

This discussion could continue on, recursively and without end. However, better it is left here, after specifying in detail a 'purer' task environment for the study of diverse problem representation.

11 References

Adelson, B. (1984). When novices surpass experts: The difficulty of a task may increase with expertise. *Journal of Experimental Psychology: Learning, Memory, and Cognition,* 10(3): 483–495.

Adelson, B. Problem solving and the development of abstract categories in programming languages. *Memory and Cognition*, 9, 422-433.

Allinson, C. W. and Hayes, J. (1996). The Cognitive Style Index: a measure of intuition-analysis for organisational research. *Journal of Management Studies*, 33: 119-135.

Allwood, C.M. (1984). Error detection processes in statistical problem solving. *Cognitive Science*, 8, 413-437.

Alpern, B. and Schneider, F.B. (1987). Recognizing safety and liveness. *Distributed Computing*, 2(3):117-126

Anderson, J. R. (1982). Acquisition of cognitive skill. *Psychological Review*, 89, 369-406.

Anderson, J. R. (1993). Problem-Solving and Learning. *American Psychologist,* 48(1), 35-44.

Anderson, J. R. and Douglass, S. (2002). Tower of Hanoi: Evidence for the Cost of Goal Retrieval. *Journal of Experimental Psychology: Learning, Memory, and Cognition*, 27(6).

Annett, J and Duncan, K.D. (1967). Task analysis and training design. *Journal of Occupational Psychology*, 41, 211-221.

Bartram, D. (1994). *PREVUE Assessment technical manual.* Hull, UK: Newland

Park Associates.

Berry, D. C., and Broadbent, D. E. (1990). The Role of Instruction and Verbalization in Improving Performance on Complex Search Tasks. *Behaviour & Information Technology*, 9(3), 175-190.

Bisant, D. and Lyle, J. (1989). A two-person inspection method to improve programming productivity. *IEEE Transactions on Software Engineering*, 15(10): 1294-1304.

Bogner, M.S. (1994). Human Error In Medicine. Hillsdale, NJ: Erlbaum.

Broadbent, D. E. (1958). *Perception and communication.* Oxford: Pergamon Press.

Broomfield, E.J. and Chung, P.W. (1995). 'Using Incident Analysis to Derive a Methodology for Assessing Safety in Programmable Systems. In F. Redmill and T. Anderson (Eds.) *Achievement and Assurance of Safety: Proceedings of the Safety-Critical Systems Symposium*, Brighton, 1995. London: Springer-Verlag. pp 223-239.

Bruner, J.S., Goodnow, J.J. and Austin, G.A. (1956) *A Study of Thinking.* New York, NY: Wiley.

Columbia Accident Investigation Board (2003). *Columbia Accident Investigation Board, Report Volume 1.* Washington, DC: U.S. Government Printing Office.

Cooke, N.M., and Schvaneveldt, R.W. (1988). Effects of computer programming experience on network representations of abstract programming concepts. *International Journal of Man-Machine Studies*, 29, 407-427.

Coury, B.G., Weiland, M.Z. and Cuqlock-Knopp, V.G. (1992). Probing the Mental Models of System State Categories with Multidimensional

Scaling. *International Journal of Man-Machine Studies* 36(5): 673-696.

Cribbens, (1987) Solid-state interlocking (SSI): An integrated electronic signalling system for mainline railway. *IEE Proceedings*, 134, 3, 148-158.

Cronbach, L. J. and Snow, R. E. (1977). *Aptitude and instructional methods.* New York, NY: Irvington.

Cullyer, J. and Wong, W. (1993) Application of formal methods to railway signalling - a case study. *Computing & Control Engineering Journal*, Feb., 15-22.

Cutler, A.N. (1991) Improving the efficiency of design checking and implementation testing - some fundamental considerations. In *Proceedings of ASPECT '91: An International Conference on Railway Control.* London, 7-9 October. Reading: Fericon Press.

Dennien, C.S. and Needle, B. (1991) Signalling design automation. In *Proceedings of ASPECT '91: An International Conference on Railway Control.* London, 7-9 October. Reading: Fericon Press.

Deutsch, J.A. and Deutsch, D. (1963). Attention: Some theoretical considerations. *Psychological Review*, 70, 80-90.

Diaper, D. (1989) *Task Analysis for Human-Computer Interaction.* Chichester: Ellis Horwood.

Eberts, R.E. and Brock, J.F. (1988) Computer-based instruction. In M. Hellander (Ed.) *Handbook of Human-Computer Interaction.* Amsterdam: Elsevier Science.

Eckhardt, D. E. and Lee, L. D. (1985). A Theoretical Basis of Multiversion Software Subject to Coincident Errors. IEEE *Transactions on Software Engineering*, 11, 1511-7.

Edwards, E., (1972). Man and machine: Systems for Safety. Proceedings of *British Airline Pilots Associations Technical Symposium*, British Airline Pilots Associations, London, 1972, pp. 2136.

Egan, D.E. (1988) Individual differences in human-computer interaction. In M. Hellander (ed.) Handbook of Human-Computer Interaction. Elsevier Science: North Holland, pp. 543-569.

Eikeseth, S., and Baer, D. M. (1997). Use of a preexisting verbal relation to prevent the properties of stimulus equivalence from emerging in new relations. In D. M. Baer and E. M. Pinkston (Eds.), *Environment and behavior* (pp. 138-144). Boulder, CO: Westview Press.

Ericsson, K.A. and Simon, H.A. (1984). *Protocol analysis: verbal reports as data.* Cambridge, MA: MIT Press

Evans, J. St.B. T., Handley, S. J., Over, D. E. and Perham, N. (2002). Background beliefs in Bayesian inference. Memory and Cognition, 30(2): 179-190.

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

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

Fewins, A., Mitchell, K. and Williams J.C. (1992). Balancing automation and human action through task analysis. In B. Kirwan and L.K. Ainsworth (Eds.) *A guide to Task Analysis*, pp. 241-252. London: Taylor & Francis.

Fitts, P.M. and Peterson, J.R. (1964). Information capacity of discrete motor responses. Journal of Experimental Psychology, 67(2): 103–113.

Foreman, K.H.D. (1988). Cognitive style, cognitive ability, and the acquisition of initial programming competence. *In* MR. Simonson and J.K.

Frederick (Eds.), *Proceedings of the Annual Meeting of the Association for Educational Communications and Technology.* New Orleans, LA: AECT.

Frese, M. and Zapf, D. (1991). *Fehler bei der Arbeit mit dem Computer. Ergebnisse von Beobachtungen und Befragungen im Bürobereich* . Bern: Huber.

G.R.J. Hockey (1997) Compensatory control in the regulation of human performance under stress and high workload: A cognitive-energetical framework. *Biological Psychology* 45(1-3), 73-93.

Gentner, D. and Stevens, A.L. (1983). *Mental Models.* Hillsdale, NJ: Erlbaum.

Gevers, W., Reynvoet, B. and Fias, W. (2003). The mental representation of ordinal sequences is spatially organized. *Cognition*, 87(3): B87-B95.

Gigerenzer, G., and Hoffrage, U. (1995). How to improve Bayesian reasoning without instruction: Frequency formats. *Psychological Review*, 102(4): 684-704.

Gomez, L. M., Egan, D. E., and Bowers, C. (1986). Learning to use a text editor: Some learner characteristics that predict success. *Human-Computer Interaction*, *2*, 1-23.

Grether, W.F. (1949). Instrument reading I: The design of long-scale indicators for speed and accuracy of quantitative readings. Journal of Applied Psychology, 33, 263-372.

Griggs, R.A. and Cox, J.R. (1982). The elusive thematic-materials effect in wason's selection task. *British Journal of Psychology*, 73: 407-420.

Guion, R. M. (1988). From Psychologists In Organizations - Special Section. *Journal of Applied Psychology*, 73(4), 693-694.

Health and Safety Commission (1991). *ASCNI Study Group on Human Factors. Second Report: Human reliability assessment - a critical overview.* London:

HMSO.

Health and Safety Executive (HSE) (1992). *The Tolerability of Risk from Nuclear Power Stations.* London: HMSO.

Hollnagel, E. (1993). The phenotype of erroneous actions. *International Journal of Man-Machine Studies*, 39(1): 1-32.

Hollywell, P. Human Dependent Failures: A Schema and Taxonomy of Behaviour. In E.J. Lovesey (Ed.) Contemporary Ergonomics. London: Taylor and Francis. 171-177.

Howes, A. and Payne, S.J. (1990). Display-based competence: towards user models for menu-driven interfaces. *International Journal of Man–Machine Studies,* 33: 637–655

Hsu, S-H. and Chan, T. (1995). A study of inspection performance and the personality of quality inspectors. *International Journal of Quality & Reliability Management*, 11(3), 55-65.

Johnson-Laird, P.N. (1983). *Mental Models.* Cambridge: Cambridge University Press.

Kahneman, D. (1973). *Attention and Effort.* Englewood Cliffs, NJ: Prentice-Hall.

Kawato, M., Gomi, H. (1992). A computational model of four regions of the cerebellum based on feedback-error learning. *Biological Cybernetics* 68: 95-103

Keele, S. W. (1968). Movement Control in Skilled Motor Performance. *Psychological Bulletin,* 70: 387-402. Discussed in Anderson, J. R. (2000). *Learning and Memory: An integrated approach.* New York, NY: Wiley.

Kelly, G. (1955). *The Psychology of Personal Constructs.* Norton.

Kirwan, B. (1994). *A Guide to Practical Human Reliability Analysis.* London: Taylor and Francis.

Kirwan, B. and Ainsworth, L.K. (1992). *A guide to Task Analysis.* London: Taylor & Francis.

Knight, J. C., and Leveson, N. G. (1986). An Experimental Evaluation of the Assumption of Independence in Multiversion Programming. *IEEE Transactions on Software Engineering,* 12(1), 96-109.

Kotovsky, K. and H. A. Simon (1990). What makes some problems really hard: Explorations in the problem space of difficulty. *Cognitive Psychology* 22: 143-183.

Kotovsky, K., Hayes, J.R., and Simon, H.A. (1985). Why are some problems hard? Evidence from Tower of Hanoi. *Cognitive Psychology,* 17(2): 248-294.

Layton, C.F. and Johnson, W.B. (1993). Job performance aids for the flight standards service. In *Proceedings of the Human Factors & Ergonomics Society 37th Annual Meeting.* Santa Monica, CA: HFES.

Leach, M. (Ed.) (1991) *Railway Control Systems.* London: Black.

Leveson, N (1995). *Safeware: System Safety and Computers.* Washington, DC: Addison Wesley

Littlewood, B. and Miller, D. R. (1989). Conceptual Modelling of Coincident Failures in Multi-Version Software. IEEE Transactions on Software Engineering, 15(12): 1596-614.

Luchins, A.S., and Luchins, E.H. (1950). New experimental attempts at preventing mechanization in problem solving. *Journal of General Psychology,* 42, 279-297.

Luger, G.F. (1976). The use of the state space to record the behavioral effects of subproblems and symmetries in the Tower of Hanoi problem, *International Journal of Man-Machine Studies*, 8: 411-421.

MacLeod, C.M., Hunt, E.B. and Mathews, N.N. (1978). Individual differences in the verification of sentence--picture relationships. *Journal of Verbal Learning and Verbal Behavior*, 17: 493--507.

Matthews, G., Jones, D.M., and Chamberlain, A.G. (1990) Refining the measurement of mood: The UWIST Mood Adjective Checklist. *British Journal of Psychology*, 81, 17-42.

Maxwell, J. Clerk. (1873). *Electricity and Magnetism.*

McKenna, F.P. (1984). Measures of field dependence: Cognitive style or cognitive ability? *Journal of Personality and Social Psychology.* 47(3): 593-603.

Miller, D.P. and Swain, A.D. (1987). Human error and human reliability. In G. Salvendy (Ed.), *Handbook of Human Factors*, New York, NY: Wiley.

Ministry of Defence (1997). *Def Stan 00-55: Requirements for Safety Related Software in Defence Equipment.* London: MOD.

Miyata, Y. and Norman, D.A. (1986). Psychological issues in support of multiple activities. In D.A. Norman and S.W. Draper (Eds.), *User Centered System Design.* Hillsdale, NJ: Erlbaum.

Moray, M. (1967). Where is capacity limited? A survey and a model. *Acta Psychologica*, 27, 84-92.

Morris, N. and Rouse, W. (1985). Review and evaluation of empirical research in troubleshooting. *Human Factors*, 27(5): 503-530.

Morrison, D.L. and Duncan, K.D. (1988). Strategies and Tactics in Fault Diagnosis. *Ergonomics*, 31(5): 761-784.

Newell, A., and Simon, H.A. (1972). *Human problem solving.* Englewood Cliffs, NJ: Prentice Hall.

Newton, I. (1687). *Philosophiae Naturalis Principia Mathematica.*

Nicola, V.F. and Goyal, A. (1990). Modeling of Correlated Failures and Community Error Recovery in Multiversion Software. *IEEE Transactions on Software Engineering,* 16: 350-359.

Norman, D.A. (1981) Categorization of action slips. *Psychological Review,* 88, 1-15

Norman, D.A. The psychology of everyday things. New York, NY: Basic Books.

Ohlsson, S. (1984a). Restructuring revisited I: Summary and critique of Gestalt theory of problem solving. *Scandinavian Journal of Psychology,* 25, 65-76.

Ohlsson, S. (1984b). Restructuring revisited II: An information processing theory of restructuring and insight. *Scandinavian Journal of Psychology,* 25, 117-129.

Ohlsson, S. (1985). Retrieval processes in restructuring: Answer to Keane. *Scandinavian Journal of Psychology,* 26, 366-368.

Pallant, A., Timmer, P. and McRae, S. (1996). Cognitive mapping as a tool for requirements capture. In S.A. Roberson (Ed.), *Contemporary Ergonomics* 1996. London: Taylor & Francis. Pp. 495-500.

Parasuraman, R., Molloy, R., and Singh, I.L. (1993) Performance consequences of automation induced "complacency". *International Journal of Aviation Psychology,* 3, 1, 1-23.

Patrick, J. (1992) *Training: Research and Practice.* London: Academic Press

Patrick, J., Spurgeon, P., and Shepherd, A. (1985) A Guide to Task Analysis: Applications of Hierarchical Methods. Aston Science Park, Birmingham: Occupational Services Ltd.

Payne, S.J. and Green, T.R.G. (1986) Task-action grammar: A model of mental representation of task languages. *Human-Computer Interaction*, 2, 2, 92-133.

Perrow, C. (1984). Normal accidents. New York, NY: Basic Books

Piso, E. (1981). Task analysis for process-control tasks. *Journal of Occupational Psychology*, 45, 247-254.

Pocius, K. E. (1991). Personality Factors in Human-Computer Interaction - a Review of the Literature. *Computers in Human Behavior*, 7(3): 103-135.

Poplin, M.S., Drew, D.E., and Gable, R.S. (1984). *Computer Aptitude, Literacy, & Interest Profile.* Austin, TX: Pro-Ed.

Popov, P. and Strigini, L. (1998). Conceptual Models for the Reliability of Diverse Systems - New Results. Presented at FTCS'28, 23 - 25 June, 1998, Munich, Germany.

Popper, K.R. (1972). *The Logic of Scientific Discovery (6th Ed.).* London: Hutchinson.

Quine, W. V. O. (1961/1953). Two dogmas of empiricism. In W. V. O. Quine (Ed.), *From a logical point of view,* pp. 20–46. New York: Harper & Row (Originally published in 1953).

Rabbitt, P.M. (1966). Errors and error correction in choice-response tasks. *Journal of Experimental Psychology*, 71: 264 –272.

Rabbitt, P.M. (1978). Detection of errors by skilled typists. *Ergonomics* (21)11: 945–958

Rasmussen, J. (1980). The human as a system's component. In H.T. Smith and T.R. Green (Eds.), *Human interaction with computers.* London: Academic Press.

Rasmussen, J. (1983). Skills, rules, and knowledge; signals, signs, and symbols, and other distinctions in human performance models. *IEEE Transactions on Systems, Man and Cybernetics*, 13, 257-266.

Rasmussen, J. (1986) *Information Processing and Human-Machine Interaction.* Amsterdam: North-Holland.

Rasmussen, J. and Lind, M. (1981). Coping With Complexity. Proceedings of the *European Conference on Human Decision Making and Manual Control,* Delft, Holland. 1981.

Reason, J. (1987). A Preliminary Classification of Mistakes. In J. Rasmussen (Ed.) *New Technology and Human Error,*. New York: J. Wiley. pp. 15-22

Reason, J. (1990). *Human Error.* Cambridge: Cambridge University Press.

Rizzo, A., Bagnara, S., and Visciola, M. (1987). Human error detection processes. *International Journal of Man-Machine Studies*, 27(5-6), 555-570.

Rizzo, A., Ferrante, D., Bagnara, S. (1994). Handling human error. In J.M. Hoc, P.C. Cacciabue and E. Hollnagel (Eds.), *Expertise and Technology: Cognition and Human Computer Interaction.* Lawrence Erlbaum Associates, New Jersey

Rouquet, J.C. and Traverse, P.J. (1986). Safe and reliable computing on board the Airbus and ATR aircraft. *SafeComp'86: Proceedings of the 5th IFAC Workshop on the Safety of Computer Control Systems.* Sarlat, France.

Rybowiak, V., Garst, H., Frese, M. and Batinic, B. (1999). Error Orientation Questionnaire (EOQ): Reliability, validity, and different language

equivalence. *Journal of Organizational Behavior*, 20, 527-547

Rycraft, H., Brown, F. and Leckey, N. (1992). Operational safety review of a solid waste storage plant. In B. Kirwan and L.K. Ainsworth (Eds.) *A guide to Task Analysis*, pp. 355-362. London: Taylor & Francis.

Sanders, M. S. and McCormick, E. J. (1993). *Human factors in engineering and design.* New York, NY: McGrawHill.

Satchell, P. (1993) *Cockpit Monitoring and Alerting Systems.* Aldershot: Ashgate.

Schneider, W. and Shiffrin, R.M. (1977). Controlled and automatic human information processing I: Detection, search and attention. Psychological review, 84, 1-66.

Senders, J.W. and Moray, N.P. (1991) *Human Error: Cause, Prediction, and Reduction.* Hillsdale, N.J.: Lawrence Erlbaum.

Shasha, D. and Lazere, C. (1998). *Out of Their Minds: The Lives and Discoveries of 15 Great Computer Scientists..* New York, NY: Copernicus.

Shryane, N.M., Westerman, S.J., Crawshaw, C.M., Hockey, G.R.J. and Sauer, J. (1998). Task analysis for the investigation of human error in safety-critical software design: A convergent methods approach. *Ergonomics*, 41: 1719-1736.

Smith, S. (1981). Exploring compatibility with words and pictures. *Human Factors*, 23, 305-316.

Sport Safety Committee of the Canadian Academy of Sports Medicine (1998). *Safety Aspects of In-Line Skating.* Gloucester, Ont: CASM.

Stroop, J. R. (1935). Studies of inference in serial verbal reactions. *Journal of Experimental Psychology*, 18, 643-662.

Swain, A.D. and Guttmann, H.E. (1983). *A Handbook of Human Reliability*

Analysis with Emphasis on Nuclear Power Plant Applications. United States Nuclear Regulatory Commission, Report NUREG/CR-1278. Washington, DC: USNRC.

Taylor-Adams, S. and Kirwan, B. (1995). Human reliability data requirements. *International Journal of Quality and Reliability Management*, 12(1): 24-46.

Tayyari, F. and Smith, J. L. (1987). Effect of music on performance in human computer interface. *Proceedings of the 31st annual meeting of the human factors society*, pp. 1321-1325. Santa Monica, CA: Human Factors Society.

Teitelbaum, R.C. and Granada, R.E. (1983). The effects of positional constancy on searching menus for instructions. *Proceedings of the Conference on Human Factors and Computing Systems 1983, Boston, MA.* New York, NY: Association for Computing Machinery, pp. 150-153.

The Victoria Climbié Inquiry (2003). A report of an inquiry by Lord Laming. London: HMSO.

Tiedeman, J. (1989). Measures of cognitive style. *Educational Psychologist*, 24(3), 261–275.

Treisman, A.M. (1964). Selective attention in man. *British Medical Bulletin*, 20, 12-16.

U.S. Department of Labor Employment and Training Administration (1982). *General Aptitude Test Battery.* Washington, DC: U.S. Employment Service.

van der Veer, G.C. (1989). Individual differences and the user interface. *Ergonomics, 32*, 1431-1449.

Vicente, K.J. and Rasmussen, J. (1992). Ecological interface design: Theoretical foundations. *IEEE Transactions on Systems, Man, and*

Cybernetics, 22, 589-606.

Vicente, K.J., Hayes, B.C. and Williges, R.C. (1987). Assaying and isolating individual differences in searching a hierarchical file system. *Human Factors*, 29(3), 349-359.

Vidulich, M.A. and Tsang, P.S. (1986). Collecting NASA workload ratings: A paper and pencil package. (Working Paper, NASA Ames Research Center, Moffett Field, CA).

Wason, P.C. and Johnson-Laird, P.N. (1972). *Psychology of Reasoning: Structure and Content.* London: Batsford.

Weiner, E.L. and Nagel, D. L. (1988). *Human factors in aviation.*. London: Academic Press.

Westerman, S.J., Shryane, N.M., Crawshaw, C.M., and Hockey, G.R.J. (1997). Engineering cognitive diversity. In F. Redmill and T. Anderson (Eds.) *Safer Systems: Proceedings of the Fifth Safety-Critical Systems Symposium*, Brighton, UK. London: Springer. pp. 111-120.

Westerman, S.J., Shryane, N.M., Crawshaw, C.M., Hockey, G.R.J. and Wyatt-Millington, C.W. (1998). A work sample analysis of safety-critical programming. *International Journal of Quality and Reliability Management*, 15(1): 61-71.

Westerman, S.J., Shryane, N.M., Crawshaw, C.M., Hockey, G.R.J., and Wyatt-Millington, C.M. (1995). Cognitive diversity: A structured approach to trapping human error. In G. Rabe (Ed.) *Safecomp '95. Proceedings of the 14th International Conference on Computer Safety, Reliability and Security*, Belgirate, Italy. London: Springer. pp. 142-155.

Wickens, C.D. (1984). *Engineering psychology and human performance.* Glenview, CA: Scott, Foresman and Company.

Wickens, C.D. (1991). Processing resources and attention. In D. Damos (Ed.), *Multiple task performance.* London: Taylor and Francis.

Wickens, C.D. and Hollands, J. (1999). *Engineering Psychology and Human Performance (3rd Edition).* New York, NY: Pearson Education.

Wilson, J.D., Nosek, J.T., Hoskin, N., Liou, L.L. (1992). The Effect of Collaboration on Problem-Solving Performance among Programmers. *Proceedings of the IFIP 12th World Computer Congress. Volume 1: Algorithms, Software, Architecture, Madrid, Spain, 1992.* Amsterdam: Kluwer, 86-93.

Wright, R.B. and Converse, S.A. (1992) Method bias and concurrent verbal protocols in software usability testing. In *Proceedings of the Human Factors Society 36th Annual Meeting.* Santa Monica, CA: HFES. pp. 1220-1224.

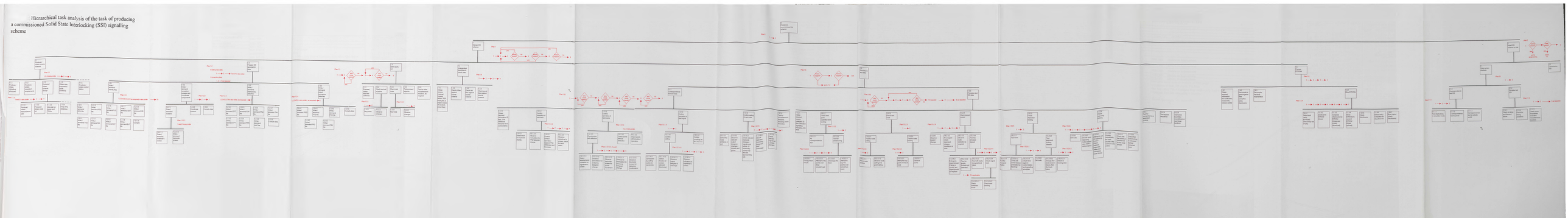
Yang, S.C. (2003). Reconceptualizing think-aloud methodology: refining the encoding and categorizing techniques via contextualized perspectives. *Computers in Human Behavior*, 19(1): 95-115.

Young, F.W. and Harris, D.F. (1990). Multidimensional Scaling: Procedure ALSCAL. In J.J. Norusis, *SPSS Professional Statistics.* 1992. Chicago, IL: SPSS Inc. pp. 396-461.

Zhang. J. (1997). The nature of external representations in problem solving. *Cognitive Science*, 21(2): 179--217

Appendix A: Hierarchical Task Analysis of producing a commissioned SSI scheme

Hierarchical task analysis of the task of producing a commissioned Solid State Interlocking (SSI) signalling scheme



Appendix B:. Training and crib sheets used for Cognitive Diversity experiments 1-3

Appendix B: Diversity experiments I-III training materials

General, 'core' training materials

1. Fault diagnosis in railway signalling software

The experimental task you will be performing is based around the job of people who search for faults in the software that controls railway signalling computer systems. Called 'Solid-State Interlocking' (SSI), these systems are installed in Britain's railways and around the world. They are used to ensure the safe and efficient operation of the railway, by managing the points, signals and so on, in the railway. When the signalmen and women operating the rail network want to allow train movements, they do so via the SSI computer, which, based on the software it contains, should not allow unsafe movements to take place. Obviously, any faults in the SSI software could affect the safe and efficient running of the railway.

Your task will be to search for faults in a PC-based simulation of the SSI computer. To enable you to do this, the following is intended as a guide, to teach you the basic elements of the railway and the signalling principles that are followed in its operation.

2. Elements of the Railway

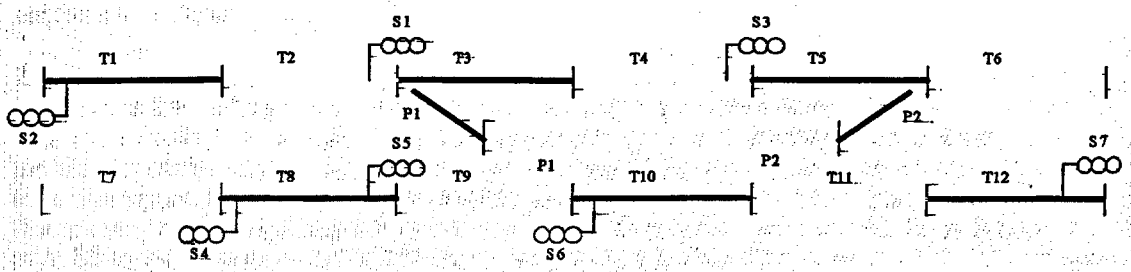


figure 1. Example railway network.

figure 1 shows an example railway network. The different elements that go to make up the railway network will now be described.

2.1. Tracks.

Railway tracks are represented on the diagram as horizontal or diagonal lines. Each line represents one railway track (i.e. the two rails that the train travels on). Figure 1 consists of two main tracks, running horizontally, with two short diagonal sections connecting them. Where the horizontal and diagonal tracks converge or diverge there is a gap, representing a set of points (described later).

2.2. Track circuits.

Each track is divided into sections called track-circuits, labelled T1, T2 etc. in figure 1 (a bit like the way the track of a model railway comes in sections). The track circuits are divided on the diagram by the short vertical lines, each one marking one of the 'ends' of the track circuit. Most track circuits are straight, with just two ends (T7, T4 etc.). A track circuit with a set of points in it (T3, T5 etc.) has three ends.

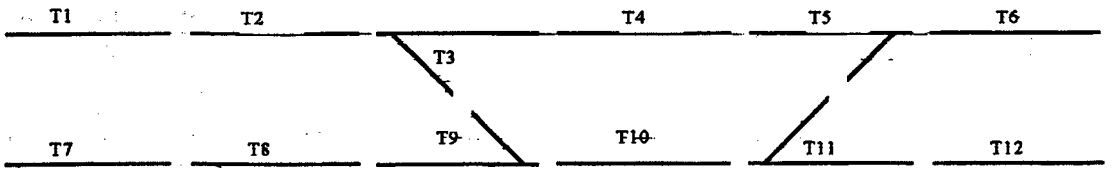


figure 2: Track circuits in figure 1.

The track sections are called 'circuits' because each one has an electrical potential between the rails, and when a train is on a particular track it completes the circuit between them. Using this information the signalling computer can detect whether there is a train in any particular track circuit. The track circuit is referred to as either 'occupied' when there is a train on it, or 'clear' when there is not. (The reason British Rail used their notorious 'leaves on the line' excuse for delayed trains was because the leaves acted as an insulator between the track and the train, and the signalling computer could not be sure where the trains were!)

2.3. Signals.

Signals (labelled S1, S2 etc. in figure 1) operate in the same way as traffic lights, in that a green light allows the train to pass and a red light means the train must stop. The signal points to either the left or the right on the diagram, and this shows in which direction routes (discussed next) are set from that signal.

2.4. Routes.

Routes are the paths across the railway network that trains travel along. Routes run between adjacent signals; the route starts from an 'entrance' signal and finishes at an 'exit signal. As mentioned earlier, the direction that the signal is pointing gives the direction of routes that are set from that signal. For instance, S1 is pointing to the right, and route 1 (R1) goes to the right. Routes only run between signals pointing in the same direction, for instance, there is no route from S5 to S4. A route is 'set' if it has been selected by the signalman, or 'not set', if it has not. There are four routes that can be set in the demonstration network, shown in the table below. (Notice that all possible routes are not listed, e.g. there is no route defined between S6 and S4)

Table 1: Routes in figure 1

Route Name	Entrance Signal	Exit signal
R1	S1	S3
R2	S1	S7
R3	S5	S7
R4	S6	S2

A route can be considered to be made up of the track circuits (and sub-routes, described later) that lie between the entrance and exit signals. These are taken from the first track circuit in line of route *after the one that the entrance signal is on*, until the one that the exit signal is on. For example, R2 runs from S1 to S7, and is made up of T3, T9, T10, T11 and T12 (shown below).

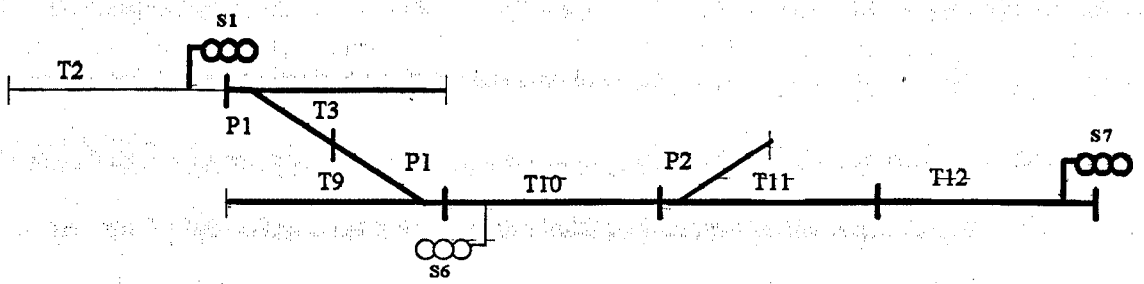


figure 3: Route 2

However, as well as R2, T3 is a part of R1 (from S1 to S3) and also R4, (from S6 to S2). To enable us to tell the computer exactly which route is set over a particular track circuit, we can describe the direction of the route over the track circuit in terms of sub-routes.

2.4.1. Sub-routes.

Sub routes correspond to track circuits, but in addition to defining a section of track, they also give the direction of the route that they are part of. For example, figure 4 below is intended to represent the two track circuits to the right of S2 in figure 1: T3 and T4.

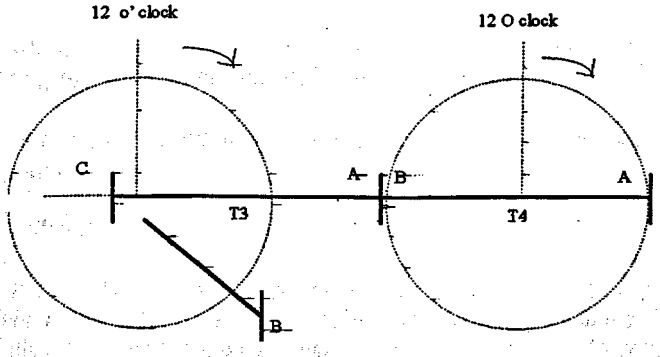


figure 4: Sub-route directions

If R1 (S1 to S3) is set, then the route is made up of T3 and T4. Considering just T4 first, we have to denote that the route is going from left to right across T4, rather than right to left. To do this, each end of the track circuit is given a label, and then the direction of the sub-route is given by specifying the labels corresponding to the ends of the track circuit that the train enters and leaves by.

The labels for the ends of the sub-route are derived by imagining a clock face laid over the track circuit (see figure 4). The imaginary clock face should be centred on the track circuit if there are no sets of points in the track circuit (i.e. T4). If there is a set of points in the track circuit, the clock face should be centred on them (e.g. T3). Starting at the imaginary 12 O'clock point and moving clockwise, each successive end of the track circuit is labelled A, B, C etc.

So, T4, above, has two ends. Clockwise from the 12 O'clock point the first end is the right-most one and this is labelled A. The next end is the left-most one and this is labelled B. Thus, the direction of the R1 sub-route over T4 is B to A, or just BA.

T3 has three ends. These are labelled using exactly the same method as before. Clockwise from the 12 O'clock point the first end is the right-most one and this is again labelled A. The next end is the bottom one (adjoining T9 in figure 1) and this is labelled B. The left-most end is again the last and is labelled C. So, the R1 subroute corresponding to T3 is labelled CA. There are two other routes that run over T3, and their corresponding sub-routes are: R2 (S1 to S7) = CB; R4 (S6 to S2) = BC.

When a route is set, all of the sub-routes that make up the route are said to be 'locked'. If a sub-route is not in a route that has been set it is said to be 'free'.

2.5. Points.

Where two railway lines meet, that is, where two tracks diverge or converge, there has to be a method of safely guiding the train down one track or the other. This is accomplished by a set of points, which physically moves the end of the track so it completes the required path.

In figure 1, points are shown as a small gap between the two adjoining lines. There are two sets of points in figure 1, labelled P1 and P2, each of which has two 'ends'. The two ends of P1 are in

T3 and T9 (both labelled P1); the two ends of P2 are in T5 and T11 (both labelled P2). Each point end can be in one of two positions, called 'normal' or 'reverse'. Normal is the position that allows a train to travel on the main, horizontal path across the points. Reverse is the position that guides the train down the diagonal path.

The two ends of a set of points always work together, i.e. they are always either both normal or both reverse. This allows trains to travel either straight along the main horizontal line, or to cross over from one main line to the other. Because they are always at the same position (normal or reverse), the railway signalling software names and treats the two ends of a set of points as though they are a single entity.

In addition to their position, the points can also be in one of two states. They can be 'free' to move from one lie to the other, or they can be 'locked' in a particular position.

These, then, are the elements that make up a railway. Next, the rules that are applied in the running of the railway are explained.

3. Railway Signalling Principles

Some rules must be applied in the working of the railway, to ensure that trains are allowed to travel across the network when it is safe to do so, but stopped when it is not. These are called 'Railway Signalling Principles', and the experiment deals with those to do with point locking and route setting.

3.2. Point locking

To allow flexibility of route setting, the points in the network should be free to move from one lie to the other whenever possible. There are two exceptions to this, however:

Track circuit locking
The points should be locked if there is a train travelling over them, as moving them would de-rail the train. SSI controls this by checking the track circuits with points in them. If they are occupied, the relevant points should be locked, and not allowed to move. Only the track circuits with points in them should do this, however. The occupation of any other track circuits should not affect the functionality of the points.

Route locking
In addition to not moving the points when a train is actually travelling over them, the high speed and momentum of trains means that the points should not be allowed to move in front of a train when it is about to cross the points. This is accomplished by the locking of all relevant sub-routes when a route is set. This has the effect of locking all the points within a route when that route is set. Only the points within a route should be locked by its setting. For instance, R1 should lock P1, but not P2.

3.1. Route Setting.

The signalling principles for route setting are based on the philosophy of allowing only one train at a time into any particular area of the network. When a signalman or woman presses a button to request a route to be set, the SSI must evaluate whether this will be the case by using the principles relating to opposing routes and signal control.

Opposing routes.
Opposing routes are those that use one or more similar track circuits, i.e. two or more routes that travel, at some point, over the same bit of track. For instance, R2 and R3 are opposing, as they both travel over T9, T10, T11 and T12. They ***should not***, therefore, be allowed to be set at the same time. Not all routes are opposing; R1 and R3 travel on parallel tracks, and at no point do they use the same track circuit. To allow the greatest functionality of the railway (i.e. the most trains running at the same time) they ***should*** be allowed to be set together.

SSI controls the setting of opposing routes by checking the status of the points within a route, to see if they are already in, or free to move to, the correct position for the route to be set (either normal or reverse); or if they are locked in the wrong position for the route.

If two routes require the same sets of points in exactly the same positions then inspecting the points will not show them to be opposing. For these 'directly' opposing routes (e.g. R2 and R4) some of the sub-routes that are set by the opposing route are also checked to make sure they are free.

Signal control

After a route is set, steps must be taken to ensure that only one train is allowed into the route at a time. This is accomplished by controlling the colour of the route's entrance signal. The signal, normally at red, must be changed to green to allow the train into the route. This is only allowed if there are no trains already in the route. SSI controls this by inspecting the state of the track circuits in the route, which should all be clear. If any of the track circuits are occupied, the entrance signal should stay red. Conversely, only the track circuits that make up a route should control the route's signal colour. If other track circuits, outside of the route, are occupied this should not stop the signal from turning green.

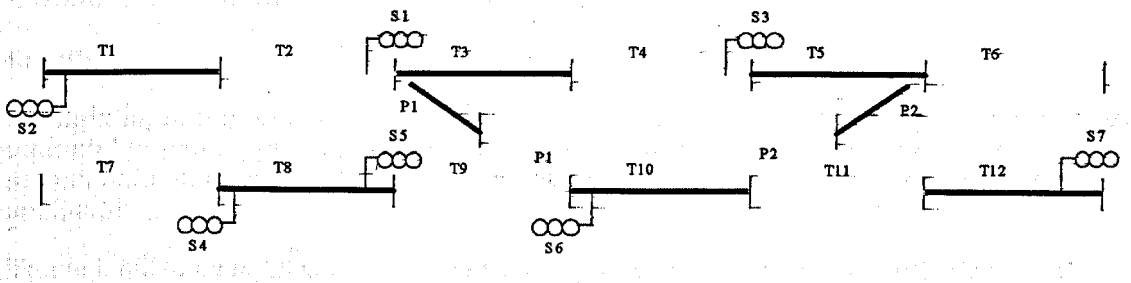


figure 1. Example railway network.

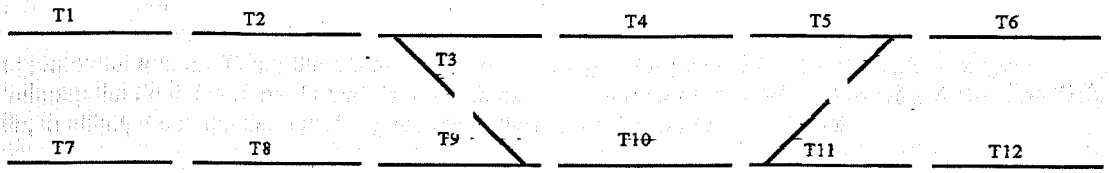


figure 2: Track circuits in figure 1.

Table 1: Routes in figure 1

Route Name	Entrance Signal	Exit signal
R1	S1	S3
R2	S1	S7
R3	S5	S7
R4	S6	S2

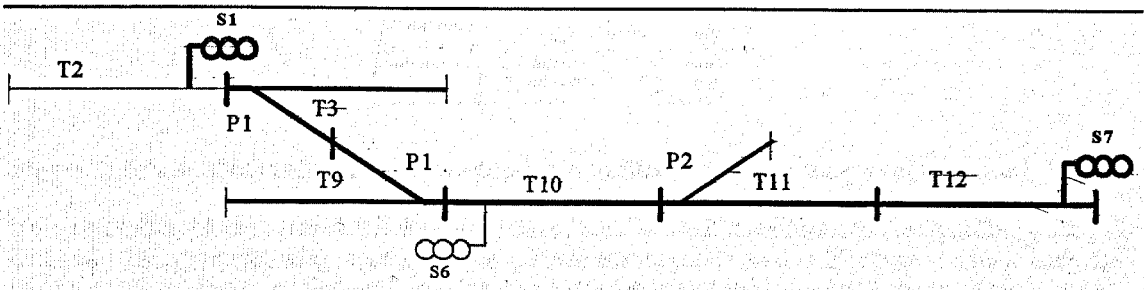


figure 3: Route 2

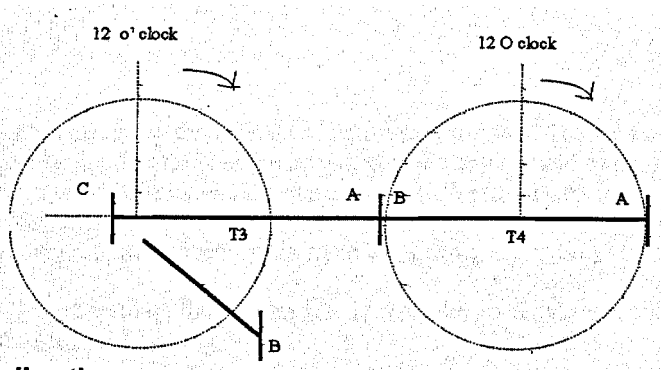


figure 4: Sub-route directions

Checkers specific training materials

Checking.

The signalling rules (described earlier) that the SSI computer operates to must first be programmed into the computer by signalling engineers. The programme must allow the maximum functionality from the rail network whilst ensuring its safety. The programme is therefore extensively checked and tested before installation.

Your task will be a simplified version of the checking procedure. You will be presented with a display showing a diagram of a section of railway. At the bottom of the diagram is a list of all the routes that are in the network. Below the diagram is a dialogue window which will display the code that will enable the signalling computer to safely operate the railway. You will be checking this code for mistakes. The code is arranged into three areas, 'PRR', 'PFM' and 'OPT', each accessed by their associated buttons.

PRR stands for 'Panel Route Request'. This means the code associated with a signalman or woman's request to set a route. When the button on the signalling panel is pressed to set a route, the computer evaluates the PRR file to check that the conditions for the setting of the particular route are met. The PRR file, in effect, contains the signalling principles for setting the routes in the layout.

PFM stands for 'Points Free to Move'. This code contains the conditions that must be met before each set of points can be moved from one position to the other.

When the PRR and PFM process a route request, the outcome is either 'true' (set the route requested) or 'false' (do not set the route). If 'true', the points are instructed to move to the position required.

OPT is short for 'OutPut Telegram'. If a 'true' value is returned by the PRR and PFM, the request is passed to the OPT file, which evaluates whether the conditions to give a green light on the entrance signal of the route are met. If this test is passed then a message is sent to the signal to change to green.

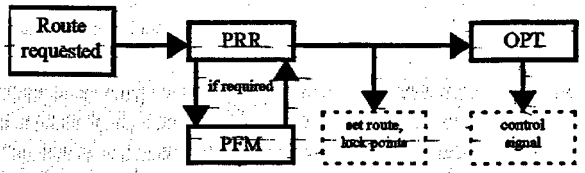


Figure 1: Schematic showing the relationship between files when processing a route request.

Your task today will be to check the data in the three 'files' to see if it conforms to the signalling principles outlined earlier. The following is intended as a guide to enable you to do this. The type of data in each file will be explained and examples used from the demonstration layout (which is **NOT** the same as the layout in the previous handout) to illustrate what you should be checking for, when you come to perform the task for 'real'.

1. PRR.

Clicking on the PRR button brings up the PRR data into the text window. The data within this is divided up into routes, with the data for the first route in the layout, then the second, and so on. Each set of route data contains the conditions required before the route can be set, and the actions that must be carried out to then set the route. The code can be scrolled through by clicking on the 'up' and 'down' buttons (one click per line). Alternatively, the data can be scrolled by the route by clicking the 'route+' button.

For an example, we will work through the data for R2. This can be accessed by scrolling through the data for R1 until all of the data for R2 (starting '*QR2 if R2 a') is visible.

The PRR route data is laid out thus:

1. Route label and availability	*QR2	if	R2 a
2. Points test			P1 crf, P2 cnf
3. Directly opposing route(s) test			U10-AB f, U3-BC f
4. Route setting		then	R2 s
5. Sub-route locking			U3-CB 1, U9-CA 1, U10-BA 1, U11-BA 1
			U12-BA 1
6. Points controlling			P1 cr, P2 cn
7. Signal clearing			S2 clear bpull

The code is divided into a set of conditions following the 'if' (lines 1 to 3) and a set of actions following the 'then' (line 4 to 7). If the 'if' conditions are passed, the 'then' actions are applied.

1.1. Route label and availability.

***QR2** is the label for the R2 route data. All routes have ***Q** at the start, which stands for 'route re**Q**uest'. This should then be followed by the route name, **R2** in this case. After the **if** comes the set of conditions that must be satisfied for the route to be set. The first of these, **R2 a** , checks that the route being requested is **a**vailable (i.e. has not been barred because of engineering work, or 'leaves on the line').

1.2. Points test.

Next, all the points in the route are tested. For R2 there are two sets of points P1 and P2. These must be reverse and normal respectively. So P1 are checked to ensure that they are **C**ontrolled **R**everse or **F**ree to be controlled reverse (crf). If the points are already at their reverse position then this test is passed. If not, then the computer inspects the PFM file (described later) to discover if they are free to be moved reverse. P2 are similarly tested, this time to see if they are **C**ontrolled **N**ormal or **F**ree (cnf). Only the points that are required by the route need to be tested. For instance, R1 (S2 to S3) does not travel over P2, so the PRR code for R1 will not include a test for P2.

1.3. Directly opposing routes test.

This part of the code is testing to see if any routes that are directly opposing to our route (R2 in this case) are set, (i.e. those routes that, between the two respective entrance signals, require the same position of the sets of points as the route we are requesting). Only the directly opposing routes need be tested for here, as other opposing routes (requiring different points positions) have already been dealt with by testing the points themselves (described earlier and in the PFM section). If there were no directly opposing routes then there would be no data here.

For R2, the only directly opposing route is R4. The other opposing routes, R1 and R3, need P1 normal. To ensure that R4 is not set we test the sub-routes that would have to be locked if R4 was set. Two sub-routes are tested for each opposing route:

1.3.a. The first corresponds to the track circuit that the opposing route's entrance signal is situated on
1.3.b. The second corresponds to the first track circuit in the route we are requesting.

In both cases we test for the sub-route direction that *corresponds to the opposing route*. So the data means:

U is the label used for sub-routes, the number corresponds to the track circuit (U10 is a sub-route across T10). The first subroute number is the same as the track circuit corresponding to the opposing route's entrance signal (T10), and the direction of the sub-route (the direction of the opposing route) is given by the letters AB (right to left). The second subroute corresponds to the first track circuit in the route we are requesting (T3), again in the direction of the opposing route, BC (bottom-right to top-left). In both cases the **f** denotes that we are testing the sub-route to ensure that it is free.

If there are no directly opposing routes, there is no data required here and line 3 should be blank. If there is a directly opposing route or routes, then for each one the two sub-routes detailed in **1.3.a+b** should be included.

1.4. Route setting
If all the conditions are met then the route can be set. The first part of this is to register the route as set, shown by the route name followed by an ' s ' for set. by the route name, then an 's'.

1.5. Sub-route locking
Next the route must be locked. This is achieved by individually locking each sub-route of the route we are setting. So, each sub-route of the route is listed, followed by an **l** for locked.

1.1.6. Points controlling
Here, the points are controlled to the position required by the route, in this case, controlled reverse (cr) for P1, controlled normal (cn) for P2.

1.7. Signal clearing
The entrance signal for the route is cleared, by writing 'clear bpull' after the entrance signal label. This requests the OPT file to check the data to clear this signal to green.

2. PFM

This is the **P**oints **F**ree to **M**ove data. As stated before, if the points are in the wrong position when a route is requested, the PFM data is consulted by the PRR to see if it is safe to move them to the required position. For each set of points in a network there will be data to check if the points can be moved normal and data to check if the points can be moved reverse. The data for **P1** is as follows:

1. ***P1N T3 c , T9 c , U3-BC f , U9-CA f**
2. ***P1R T3 c , T9 c , U3-CA f , U9-BA f**

2.1. P1N

P1** specifies that this is the PFM data for **Points 1,** showing the conditions required to move the points Normal **(P1N).** The asterisk '**' is the label used for points in the PFM data.

T3 c , T9 c
First the track circuit(s) that the points are in are tested to make sure they are clear. It is obviously not very safe to move the points with a train moving over them.

U3 B-C f , U9 C-A f
The programme then looks to see if any routes have been set over the points which would require them in the opposing direction to the way that we wish to move them, in this case normal. This has the effect of testing for sub-routes from opposing routes that require the points to be at the opposite position as the way we want to move them.

For P1N, two routes require that the points be Reverse: R2 (from S2 to S6) and R4 (from S5 to S1). Rather than test for all of the sub-routes in those routes, the code just tests the last sub-route in line of route over the points.

The sub-routes set by R2 across the points are U3-CB and U9-CA. Only the last sub-route, in the direction the train would be travelling, is needed. As R2 goes from S2 to S6, a train on this route crosses U9-CA last, so only this needs inclusion.

The other route that requires P1 reverse is R3, from S5 to S1. The two sub-routes in this case are the ones describing the reverse directions to the above, namely U9-AC and then U3-BC. Again, only the last one is required, U3-BC. Both sub-routes are followed by an ' f' ', as they are tested to see if they are 'free'.

2.2 P1R

For P1R, the same track circuits are checked as for P1N. This time, though, we must test the sub-routes over the points of those routes that need P1 to be normal. Those routes are R1 and R3. R1 sets U3-CA and R3 sets U9-BA, so these are the sub-routes that are tested.

3. OPT data.

3.1 *S2 R2 T3 c , T9 , T10 c , T11 c , T12 c

The PRR and PFM data check whether it is safe to set a route. If it is then the request is passed to the OPT which checks that there are no trains in the route before sending the OutPut Telegram to the signal for the route, telling it to turn green. The first part of the OPT data will be a '*' followed by the entrance signal for the route. Next, because there can be more than one route set from a signal, will be the route name. Following this, all the track circuits in the route are listed, each with a 'c' after it, signifying we are testing it to see if it is clear. Each route should have a separate line of OPT data.

4. Fault reporting.

The purpose of this exercise is to find any errors that may be present in the code that you will check. If you do find an error then report it in the following manner:
Click on the button marked 'log error' in the bottom right-hand corner of the screen. A cursor will now appear in the box at the bottom of the screen. You can type in up to two lines of text to specify the error that you found, pressing return at the end of the first line to get onto the second.

For each error you find, include the following information:

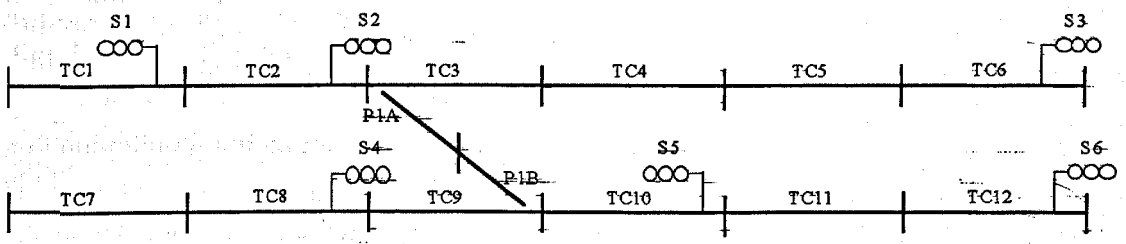
The file you are in (PRR, PFM or OPT)
The part of the file (Route name for PRR and OPT, Point name and direction for the PFM)
The specifics of the error, i.e. **what** is wrong, and **why** it is wrong.
Whether the error is one of safety or of functionality

For example:

PRR, R4, 'R4 s' is missing after the 'then'. Safety
or
PFM, P2R, subroute 'U5-BC' is wrong, should set 'U5-BA' instead. Safety
or
OPT, R1, T12 included unecessarily. Function

When you have finished typing, just click on the 'log error' button again. This saves your text, and you can continue checking for more errors. If you start typing and then realise that you have made a mistake (i.e. there is no error where you thought there was), just press the 'Esc' key. This will return you to checking, but not save the contents of the error window.

Reference Guide



Entrance Signal	Exit signal	Route Name
S2	S3	R1
S2	S6	R2
S4	S6	R3
S5	S1	R4

1. Conditions to set a route:

A) The points must be set to the correct position, or free to be set.
B) No opposing routes must be set.
C) No trains must already be in the route.

1.1. Actions when the conditions have been passed:

D) All the sub-routes are locked
E) The points are controlled to the required position

2. The signalling code specifying the above for R1B looks like:

R1B

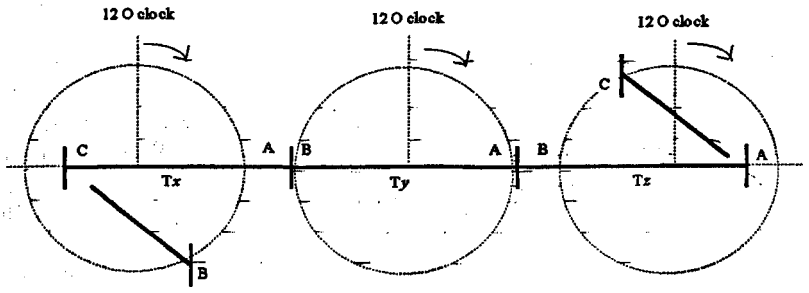
if P1 crf
U3 B-C f
T3 c , T9 , T10 c , T11 c , T12 c
then set R1B
U3 C-B l, U9 C-A l , U10 B-A l, U11 B-A l, U12 B-A l
P1 cr

Points Free to Move

P1N T3 c, T9 c, U3 B-C f, U9 C-A f

Track circuits	**T**[*x*] , **T**[*y*] , **T**[*z*]
Sub-routes	**U**[*x-AB*] , **U**[*y-AB*] , **U**[*z-AB*]
Points	**P**[*x*] , **P**[*y*] , **P**[*z*]
Signals	**S**[*x*] , **S**[*y*] , **S**[*z*]
Routes	**R**[*x*] , **R**[*y*] , **R**[*z*]

Sub route direction given by:



PRR (Panel Route Request)
One block such as this per route

1. Route label and availability	***Q**[*route number*] if	**R**[*route number*] **a**
2. Points test		**P**[*x]* {***cnf*** (normal) or ***crf*** (reverse)} , **P**[*y*] ...
3. Directly opposing route(s) test (if required)		**U**[*x-AB*] **f**, **U**[*y-AB*] **f**
4. Route setting	then	**R**[*route number*] **s**
5. Sub-route locking (all sub-routes in the route)		**U**[*x-AB*] **l**, **U**[*y-AB*] **l**, **U**...
6. Points controlling		**P**[*x]* {***cn*** (normal) or ***cr*** (reverse)} , **P**[*y*] ...
7. Signal clearing		S[entrance signal] **clear bpull**

PFM 'Points Free to Move'
One block such as this per set of points

***P**[x]**N** **T**[x] **c**, ...(track circuits for P[x]) **U**[*x-AB*] **f**, ...(last sub-route over the points for each route that requires the points reverse)

***P**[y]**R** **T**[x] **c** ...(track circuits for P[y]) **U**[*x-AB*] **f**, ...(last sub-route over the points for each route that requires the points normal)

OPT 'OutPut Telegram'
One block such as this per route

***S**[*x*] **R**[*x*] **T**[*x*] **c**, **T**[*y*] **c**, **T**[*z*] **c**, ...(all the track circuits in the route)

Testers specific training material

Testing.

The signalling rules (described earlier) that the SSI computer operates to must first be programmed into the computer by signalling engineers. The programme must allow the maximum functionality from the rail network whilst ensuring its safety. The programme is therefore extensively checked and tested before installation.

Your task will be a simplified version of the testing procedure. This testing is carried out on a simulator of the SSI computer. The code that the signalling engineers have written for the rail network is used by the simulator, which mimics the behaviour of the real railway. Conditions can be set up and tested on it to ensure that the railway is conforming to the signalling principles discussed earlier.

1. Testing simulator.

The simulator consists of a diagram of the railway network in the upper half of the screen, and a collection of buttons and indications in the lower half of the screen (please note that the example layout shown is **NOT** the same as the layout in the previous handout). The routes in the network are defined in a route list under the diagram.

1.1. Track circuits.

The row of buttons at the bottom, labelled T1, T2 etc. are the buttons for the track circuits, which are similarly labelled on the diagram. When the track circuit is clear, i.e. there is no train on it, the button is green. The button can be clicked on to simulate a train occupying the track circuit. In this situation the button turns red.

1.2. Signals.

The red circles represent the signals in the layout. They cannot be clicked on, but instead they show the colour that their associated signal is showing, either green or red.

1.3. Points.

Each set of points on the diagram has two blue buttons associated with it. The left-most button controls the position of the points, and shows an N when the points are normal, R when they are reverse. The right-hand button shows whether the points have been 'keyed' by the signalman (K) or are free (F). If the points have been keyed, then their position, either normal or reverse, is locked, and they cannot be moved until they are un-keyed, or made free. If they are free then they can be moved between normal and reverse by clicking on the position button.

The points position can be changed by the simulator. For instance, if a route button is pressed and all the conditions are met for the route to set, then the points will be moved to their correct position for that route. Only the tester can key (lock) the points, however.

1.4. Routes.

The buttons at the top left are for setting the routes in the network. The entrance and exit signals for each route are given in a list just below the diagram. When a route button is clicked on the simulator evaluates the signalling principles programmed into it, and if they are met then the route is set. If a route is set, then the route button should change from red to green. Additionally, the entrance signal of the route should also change to green (unless one of the track circuits in the route is occupied, in which case the signal should stay red).

2. Testing procedure.

Your task will be to test the network, ensuring that points are locked or free, and routes will set, only when the signalling principles outlined earlier are met. The tests that need to be carried out will be described below. While testing, however, it is worth bearing in mind the following three areas, which are what the tests ultimately boil down to:

Points. To set a route the points must either be already at the correct lie, or be free to move to the correct lie; when the route sets only the points required by the route should be altered; and when set in a route or with their track circuits occupied they shouldn't be moveable until the route is un-set or the track circuit is no longer occupied. However, the points should be free to move at all other times

Routes. To set a route, no opposing routes must already be set. If two routes are not opposing, they should be settable at the same time.

Trains. Before the signal for a route will change to green, there must be no occupied track circuits in the route. If a route is set with any of its track circuits occupied, the signal should stay red. Track circuits should only influence the route that they are part of, and if occupied, they should not stop the signals of any other routes from turning green

To ensure that the above holds true for the network, the following tests should be performed:

2.1. Preliminary tests.

Are all the buttons present and correctly labelled? Each button must initially be operated to ensure that it is working properly.

2.2. Track circuit locking.

With no routes set, occupying the track circuit with a set of points in it should lock those points. That is, when the track circuit button corresponding to the track circuit over a set of points is pushed, simulating a train occupying the track circuit, the points should not be moveable from normal to reverse or vice versa, regardless of the 'keyed' button position.

2.3. Point moving and locking. These tests ensure that the points are moved correctly by routes, and are then locked .

2.3.1. Individually, each set of points in a route should be moved, and keyed, to conflict with that route. For example, R2 needs P2 to be normal, then P2 moved reverse and keyed. R2 should not be settable. This test should be done individually for each set of points in a route.

2.3.2. Conversely, each route should be set, and each set of points within the route individually tested to ensure they are locked (cannot be moved).

2.3.3. With no routes set, move all points to normal.. Set a route. Only the points required reverse by the route should move to reverse. Repeat with points initially set to reverse.

2.4. Train in route locking.

2.4.1. To ensure that each track circuit in a route will stop the entrance signal from changing to green if it is occupied, occupy a track circuit in that route and then set the route. The route should set, but the signal should stay at red.

2.4.2. Conversely, each track circuit not in a particular route should have no affect on the entrance signal to the route from changing to green.

2.5. Opposing locking.

2.5.1. With a route set, ensure that all opposing routes cannot be set.

2.5.2. With a route set, ensure that all routes that are **not** opposing routes **can** be set.

3. Error reporting.

The purpose of this exercise is to find any errors that may be present in the behaviour of the network that you will subsequently test. If you do find an error then report it in the following manner:

Click on the button marked 'log error' in the bottom right-hand corner of the screen. A cursor will now appear in the box at the bottom of the screen. You can type in up to two lines of text to specify the error that you found, pressing return at the end of the first line to get onto the second.

For each error you find, include the following information:

The elements of the railway under test, and their state (e.g. P1 locked, T3 clear, R4 set)
The specific functionality that is not correct, i.e. **what** is wrong, and **why** it is wrong.
Whether the error is one of safety or of functionality [**e.g. for experiment III**]

For example:

T3 occupied, R4 set, S4 does not change to green when it should. Function

R4 set, can also set opposing R6. Safety

P1 keyed N, R4 will set when it shouldn't. Safety

When you have finished typing, just click on the 'log error' button again. This saves your text, and you can continue checking for more errors. If you start typing and then realise that you have made a mistake (i.e. there is no error where you thought there was), just press the 'Esc' key. This will return you to checking, but not save the contents of the error window.

Appendix C: Mental Model questionnaires.

Appendix C: Mental model questionnaires

Conceptual distance questionnaire

This questionnaire is intended to gauge how similar or different you perceive the various elements and states of the railway to be.

Each item contains a pair of constructs on the left hand side. Each construct consists of an element of the railway, e.g. points, track circuits, and one of the two possible 'states' it could be in, i.e. points can be 'free' or 'locked', track circuits can be 'clear' or 'occupied'.

Specifically, we'd like to find out, in your opinion, what the 'conceptual distance' between the two constructs is.

For instance, the first item pairs 'points free' and 'points locked' together. You may believe that, because both constructs are dealing with points, the conceptual distance between them is very low (i.e. they are both very similar). In that case, you'd mark the scale line near to the left hand, zero, end of the scale.

Alternatively, you may think that because the constructs refer to very different states of the points, 'free' and 'locked', that they are very conceptually distant (i.e. dissimilar). In that case, you'd mark the scale line towards the right-hand, 100, end of the scale.

For each item, please make one mark on the scale indicating how conceptually distant or similar you think each pair of constructs are. There are no 'right' or 'wrong' answers, it's your opinion that we're after. Don't think too long about each answer, it's your first impression that is often the best.

The results of this questionnaire will be kept confidential, so please remember to write down only your participant number in the space provided over the page.

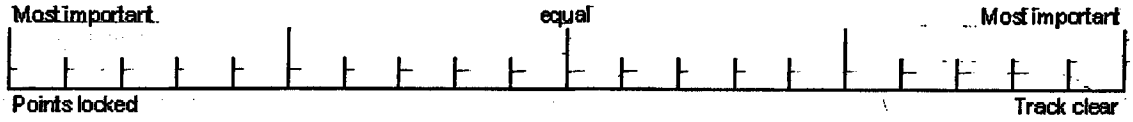
'Safety' questionnaire

This questionnaire is intended to gauge relatively how much you perceive the various elements and states of the railway contribute to its safe operation.

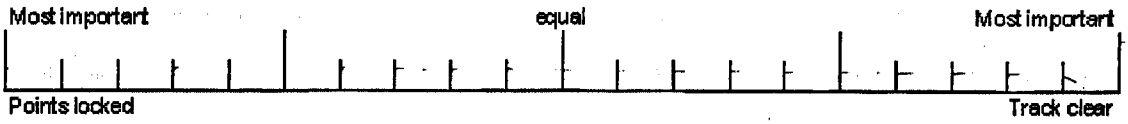
Each item contains a pair of constructs below the scale, one on the left one on the right. Each construct consists of an element of the railway, e.g. points, track circuits, and one of the two possible 'states' it could be in, i.e. points can be 'free' or 'locked', track circuits can be 'clear' or 'occupied'.

Specifically, we'd like to find out, in your opinion, which one of the two constructs contributes most to the safe running of the railway. Note that we'd like to know for each item, **the relative contribution to safety of the one construct compared to the other**, not the overall amount of safety that the constructs contribute to the safety of the railway.

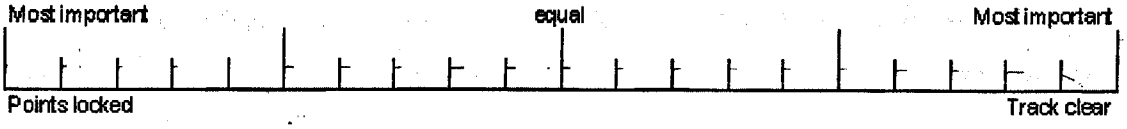
For instance, one of the items pairs 'points locked' and 'tracks clear' together. You may believe that 'points locked' contributes more to the safety of the railway than 'tracks clear'. In that case, you'd mark the scale line towards the left hand end, showing that 'points locked' is most important in terms of safety.



Alternatively, you may believe that 'tracks clear' contributes more to the safety of the railway than 'points locked'. In that case, you'd mark the scale line towards the right hand end, showing that 'tracks clear' is most important in terms of safety.



You might instead think that both constructs contribute approximately equally to the safety of the railway. In that case, you'd mark the scale towards the centre.



For each item, please make **only one mark** on the scale indicating which construct of the two you feel contributes most to the safety of the railway. There are no 'right' or 'wrong' answers, it's your opinion that we're after. Don't think too long about each answer, it's your first impression that is often the best.

The results of this questionnaire will be kept confidential, so please remember to write down only your participant number in the space provided over the page.

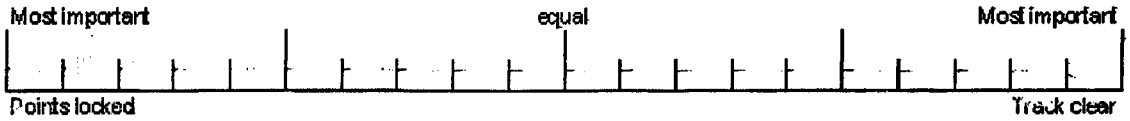
'Functionality' questionnaire

This questionnaire is intended to gauge relatively how much you perceive the various elements and states of the railway contribute to its functionality.

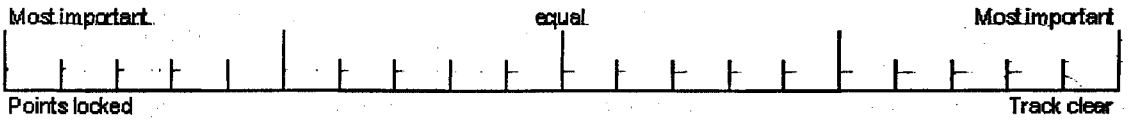
Each item contains a pair of constructs below the scale, one on the left one on the right. Each construct consists of an element of the railway, e.g. points, track circuits, and one of the two possible 'states' it could be in, i.e. points can be 'free' or 'locked', track circuits can be 'clear' or 'occupied'.

Specifically, we'd like to find out, in your opinion, which one of the two constructs contributes most to the functionality of the railway. Note that we'd like to know for each item, **the relative contribution to functionality of the one construct compared to the other,** not the overall amount of functionality that the constructs contribute to the functioning of the railway.

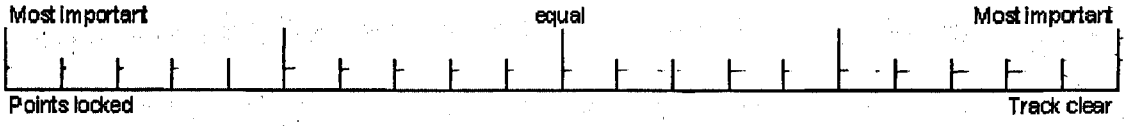
For instance, one of the items pairs 'points locked' and 'tracks clear' together. You may believe that 'points locked' contributes more to the functionality of the railway than 'tracks clear'. In that case, you'd mark the scale line towards the left hand end, showing that 'points locked' is most important in terms of functionality.



Alternatively, you may believe that 'tracks clear' contributes more to the functionality of the railway than 'points locked'. In that case, you'd mark the scale line towards the right hand end, showing that 'tracks clear' is most important in terms of functionality.



You might instead think that both constructs contribute approximately equally to the functionality of the railway. In that case, you'd mark the scale towards the centre.



For each item, please make **only one mark** on the scale indicating which construct of the two you feel contributes most to the functionality of the railway. There are no 'right' or 'wrong' answers, it's your opinion that we're after. Don't think too long about each answer, it's your first impression that is often the best.

The results of this questionnaire will be kept confidential, so please remember to write down only your participant number in the space provided over the page.