An Evaluation and Comparison of PLC Programming Techniques

Vivek Hajarnavis

Innovation Report

Submitted in partial requirement for the Degree of

Doctor of Engineering

THE UNIVERSITY OF
WARWICK

Department of Engineering
University of Warwick
August 2006

Abstract

Few significant changes in Programmable Logic Controller (PLC) software design techniques have taken place since PLC's were first introduced in the 1960's. Programs written in the traditional language used in PLC's, ladder logic, are generally thought to be difficult to maintain and modify, and thus ill suited to the support of modern flexible manufacturing processes.

This work demonstrates that the choice of PLC software structure used in a project has an impact on process flexibility with an appropriate choice providing significant cost savings in development time.

An overview of work on formalised programming tools conducted in academia is provided together with a report on the PLC software structures used in industry. The factors influencing the choice of PLC and software structure are identified. Familiarity was found to be a major factor influencing selection. A method for comparing code structures, which allows the results to be expressed as a time saving (and consequently a cost) has been created. Implementation of this approach was used to show that the formalised programming tool under test provides a 33% increase in "right first time" rate together with an 80% time saving over traditional contact based ladder logic. Among experienced practitioners, performance with step-based ladder logic was found to be a close match to the formalised tool, demonstrating that the commonly perceived limitations are the result of the structure in which the language is used rather than a function of the programming tool itself.

Further investigation of participant preferences among skilled PLC users showed a mismatch between their performance with a tool and their preference, with at least 25% selecting a tool based on their prior knowledge rather than performance. This highlights the need for the use of objective measures when conducting evaluations between products and technologies.

With the information provided in this work, automation end users are provided with a mechanism for ensuring the selection of automation tools best suited to their business needs, whilst at the same time providing automation vendors with the ability to best demonstrate the strengths of the products.

Declaration

The work presented has been undertaken by myself unless otherwise acknowledged.

This work has not been previously submitted for any other award.

August 2006

University Of Warwick

Acknowledgements

I would like to acknowledge the invaluable guidance provided by my mentors, Dr. Ken Young of the University of Warwick and Mr. Ray Daniels of Rockwell Automation over the course of completing this Engineering Doctorate.

I would also like to extend my appreciation to the many people at Rockwell Automation who have provided helpful guidance and assistance over the past few years, especially Phill Hannam and Peter Vorley, and also to thank all the other industrial collaborators for their willingness to spare time to answer my questions and participate in experimental work.

Finally, I would like to extend my thanks to Graham Bing and Rodrigo Zapiain for their support, advice and good humour during my time at the University at Warwick.

Contents

List of Figures

List of Tables

List of Abbreviations

EC = Enterprise Controls
HMI = Human Machine Interface
IEC = International Electrotechnical Commission
PLC = Programmable Logic Controller
RFT = Right First Time
SFC = Sequential Function Chart
UML = Unified Modelling Language
XML – Extensible Markup Language

1.0 INTRODUCTION

1.1 BACKGROUND

Programmable Logic Controllers (PLC's) are microprocessor-based computers designed for the implementation of control algorithms in industrial environments. Originally designed to replace hard-wired relay-based machine control systems in the 1960's, PLC's remain popular to date owing to their reliability, simplicity and guarantees of long-term support from vendors. Owing to the origin of PLC's and need for the controllers to be understood by the electricians who had previously worked on hard-wired control systems, a graphical programming language called ladder logic was developed. Ladder logic remains the dominant language for programming PLC's, even though several other options exist: Sequential Function Chart (SFC), Instruction List and Structured Text. These languages are outlined in IEC61131, a multi-part international standard encompassing various aspects of using and applying PLC's, providing general information about terminology, defining languages and giving guidelines for the application and implementation of the respective languages. (Lewis, 1998).

Increases in processor power, coupled with advances in the PC-based programming tools used to configure programmable controllers has opened up new possibilities for end users of automation products to implement logic control algorithms. However, it is well recognised that industrial automation users are conservative in nature with practice in many factories little changed since the advent of the first PLC's. In order to encourage change in industry, automation vendors are therefore in need of a better understanding of their customers'

requirements, together with a new mechanism for demonstrating the strengths of their products. This research focuses on delivery of these aims.

1.2 PROJECT MOTIVATION AND OBJECTIVES

Motivation for this project derived from implementation work conducted in the preliminary stages of the project in which software tools were used to create control software to operate a manufacturing cell at the University of Warwick, shown in Figure 1. This work was conducted in order to create an automation demonstration and test facility and consisted primarily of the deployment of SFC and three versions of a commercial formalised programming tool called Enterprise Controls (EC). Differences in ease of use were perceived with each of the tools, inspiring the idea of capturing programming tool effectiveness in an objective and rigorous manner.

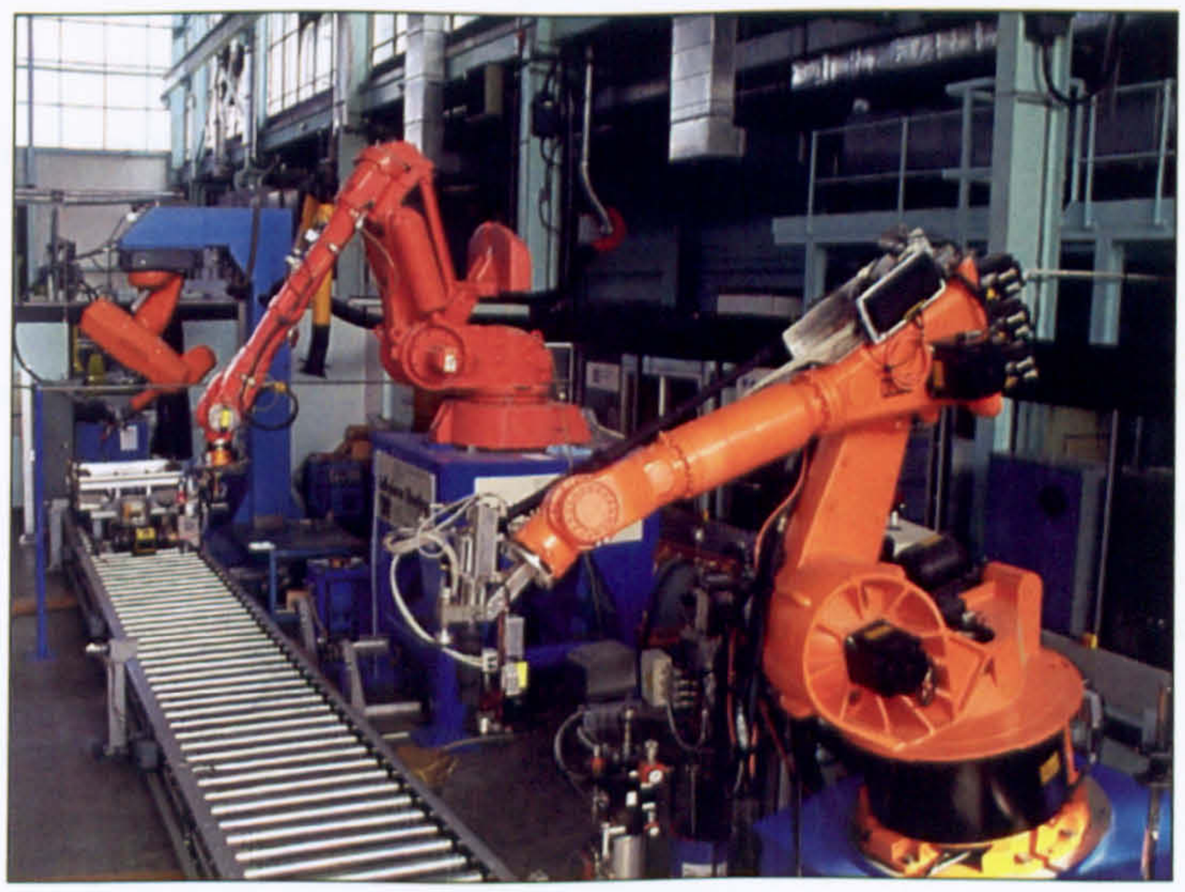

Figure 1 Future Automation Control Technology Cell

Based on the experience gained from this work, and an awareness of the conservative nature of automation users in the car industry, the main objective of this research project was defined to be the expression of the impact of PLC logic design methodology choice in terms of business benefit. In order to achieve this, a set of tasks was identified: the establishment of how PLC's can be programmed, to look at which languages and code structures are most common in the automotive industry at present, to gain an indication of geographic preferences and to use this information to quantify the business benefits of using one of these programming techniques over the others. Following on from this, a further aim was to measure the impact of prior experience on the choice of logic design methodology.

The justification for this work was based on the idea of conducting an objective comparison in order to provide information to encourage end users to consider programming methods other than those used at present.

1.3 PORTFOLIO STRUCTURE

The structure for this portfolio reflects that of this innovation report and individual portfolio submissions can be seen as the individual chapters which when read together form a complete thesis. The order in which submissions should be read is as follows:

- Portfolio submission 2 – literature review
- Portfolio submission 3 – industrial survey
- Portfolio submission 4 – experiment proposal
- Portfolio submission 5 – experiment application, analysis and results

- Portfolio submission 6 – published material

Portfolio submission 1 describes the results of an application which followed from the initial risk assessment of the facilities at Warwick. This is an interesting result in its own right, and conducting this work provided exposure to function block programming in Pilz safety system processors. Portfolio submissions 2-5 provide additional details to support the information presented in chapters 2.0 to 5.0 respectively.

2.0 REVIEW OF PROGRAMMING TECHNIQUES IN INDUSTRIAL CONTROL SYSTEMS

Background information which helped formulate the project objectives and its justification, as well as the methodology by which it has been achieved is presented in portfolio submission 2. The report describes and reviews previous work in the area of PLC programming and the creation of logic control systems, outlining the strengths and the limitations of existing research and uses this to help define the main research question. In doing so, it provides in-depth analysis at the outset of the doctorate.

The report describes the options available to a user for programming PLC's, starting with an outline of the languages defined in the IEC61131 standard and then going on to describe formalised programming techniques and comparison work conducted between the respective ideas. An overview of these findings is provided in this chapter.

2.1 PLC PROGRAMMING TECHNIQUES

2.1.1 The IEC61131 Standard

IEC61131 is a multi-part standard encompassing various aspects of using PLC's in control applications, providing general background information, defining languages and giving basic guidelines for their application and implementation. A key feature of the standard is that it aims to address the deficiencies of conventional ladder logic through encouraging well structured "top-down" or "bottom-up" program development, strong data typing, full execution control,

support for the realisation of complex sequential behaviour, support for data structures, flexible language selection and vendor independent software elements (Lewis, 1998). Although the standard was created in order to aid the standardisation of PLC programming tools supplied by different vendors, it is thought to lack clarity and is open to interpretation (Öhman *et al*, 1998). In order to address this and achieve the aim of portable control software, a number of companies formed a trade association called PLCopen in 1992, which aims to define compliance levels to the standard. Products which attain a specific compliance level will support a known level of software portability. Despite these limitations, the standard provides a useful starting point for gaining awareness of methods for programming PLC's.

The third part of the standard, IEC61131-3 defines five programming languages: ladder logic, sequential function chart (SFC), instruction list, function block and structured text. The function block programming language is of particular interest to many practitioners as it provides a mechanism for the encapsulation of industrial algorithms in a form which can be understood by people who are not software specialists. A second international standard, IEC61499, defines how function blocks can be used in industrial process applications (Lewis, 2001). However, the description of languages provided here is limited to ladder logic and SFC as they are most relevant to the work described in this report. Further information, including details of the other languages can be found in submission 2 and (Lewis, 1998).

2.1.1.1. Ladder Logic

Historically, control logic software in Programmable Logic Controllers (PLC's) has been written using ladder logic, a graphical programming language which represents the electrical systems used for control purposes before microprocessor-based control systems came into common use. Ladder logic is a graphical representation of the "if...then" construct used extensively when programming with traditional text-based structured computer programming languages. Inputs, represented by switches (back-to-back square brackets) can be combined to form Boolean expressions and then related to outputs, represented by coils (parenthesis), as shown in Figure 2 and Figure 3 which show two different structures in which the language can be used. Ladder programs are generally analysed left to right and top to bottom although this is dependent on the specific PLC and associated programming tool. The visual resemblance of the code to a ladder gives the programming language its name. As well as simple operations for manipulating bits of input and output (I/O) data, programmers can also make use of more sophisticated functions allowing the creation of timers and counters. Functions are also available for handling data words, arrays of data and mathematical operations.

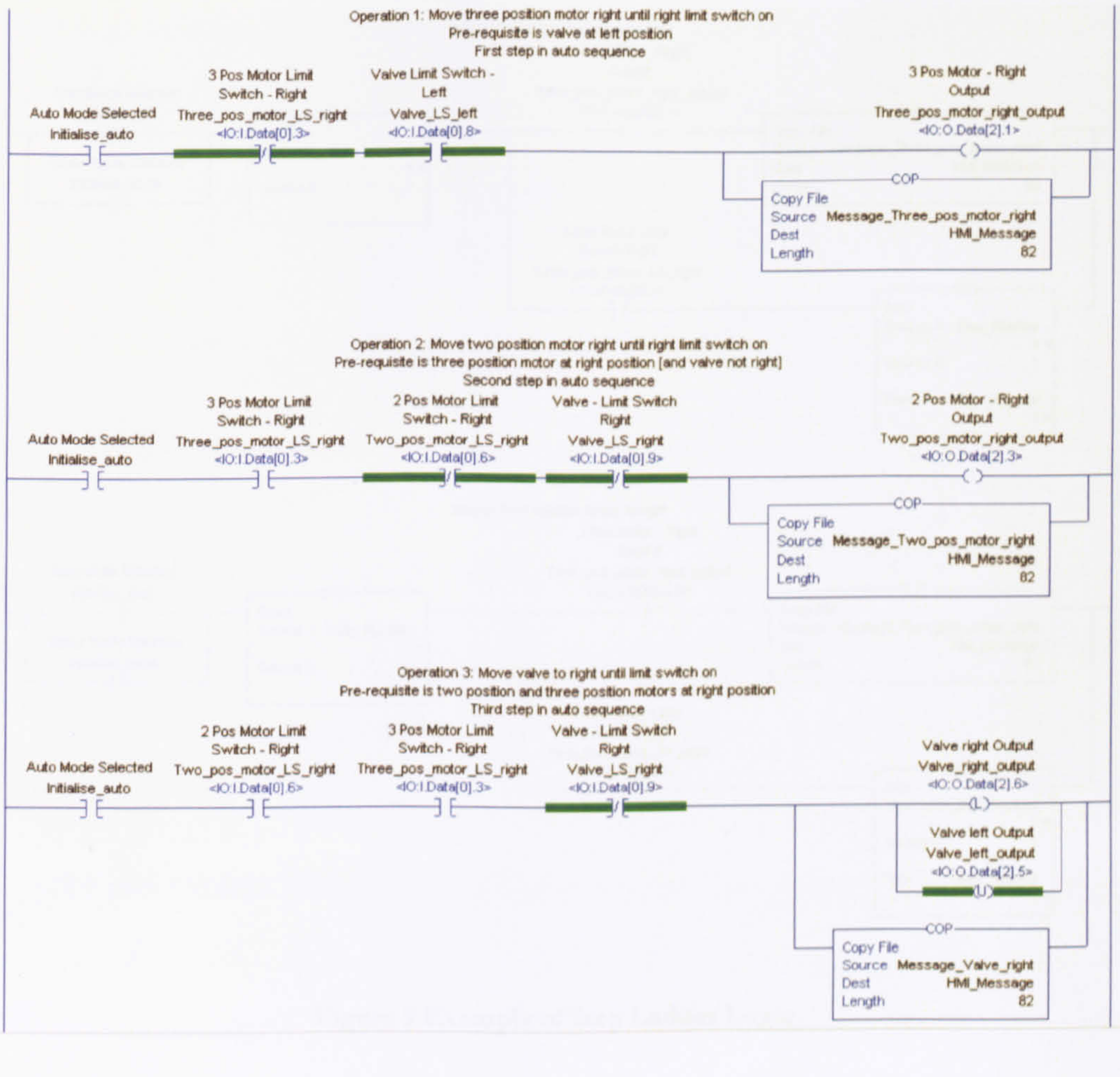

Figure 2 Example of Contact Ladder Logic

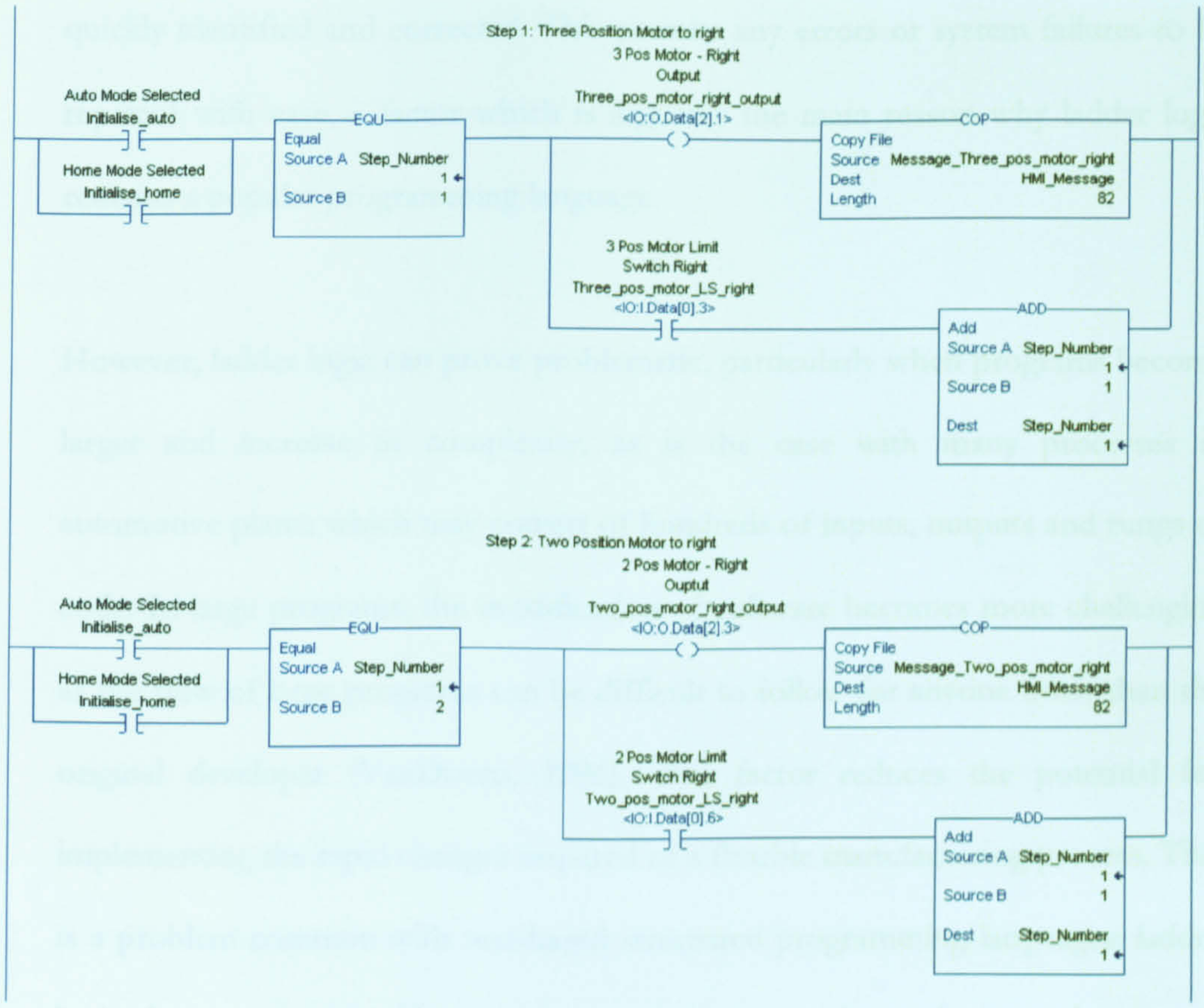

Figure 3 Example of Step Ladder Logic

The instructions in the program tend to be scanned by the processor in a sequential order, although "jump" commands can be used to modify the order in which operations take place if necessary.

Ladder logic has advantages owing to its simplicity and ease of diagnostics. Visual display of the control program in the form of an electrical wiring diagram means that it is easy to identify whether the status of an individual bit of data is in the correct state. Data bits in the PLC program correspond to hardware input or output points, or to internal registers holding information concerning the control process. In simple programs, the ability to access this low level of information serves as a very powerful diagnostic tool; programmers and maintenance technicians can view input or output states and any incorrect conditions can be

quickly identified and corrected. This permits any errors or system failures to be repaired with ease, a factor which is arguably the main reason why ladder logic remains a popular programming language.

However, ladder logic can prove problematic, particularly when programs become larger and increase in complexity, as is the case with many processes in automotive plants which may consist of hundreds of inputs, outputs and rungs of code. In large programs, the modification of software becomes more challenging as the flow of large programs can be difficult to follow for anyone other than the original developer (VanDoren, 1996). This factor reduces the potential for implementing the rapid changes required of a flexible manufacturing process. This is a problem common with text-based structured programming languages: ladder logic does not lend itself to consistency in programming and reuse of code and two programmers writing software to operate a piece of machinery may produce very different solutions.

Beyond a point, the diagnostic capabilities of ladder logic reach a limit. Tracking values of data words is more challenging than reading the status of data bits, particularly if the data word of interest is changing rapidly. Similarly, tracking of timing and transient issues can be difficult in ladder logic. In this case, special tools are required in the programming software to allow monitoring of trends over an extended period. A further weakness of ladder logic is that it does not lend itself to reuse of code and functionality. Most tools for programming ladder logic provide the opportunity to copy and paste functionality from previous work. Whilst effective, this approach is prone to errors, particularly if the copied code is

not adapted properly. A variable left unchanged can cause serious disruption to the control operation and prove difficult to diagnose.

2.1.1.2. SFC

SFC was derived from Grafcet, a graphical language based on a French national standard (now a European standard, EN60848) and itself an evolution of a Petri-net, an academic tool used for modelling and describing control software and manufacturing systems (David, 1995). Rather than being a language in its own right, SFC can be seen as a method for organising programs, allowing large programs to be broken up into smaller, more understandable sections. SFC's consist of step and transition pairs, as shown in Figure 4. Steps are depicted by rectangles and transitions by horizontal lines. Code written in ladder logic, structured text or enclosed in a function block is associated with each step and transition, and the principle is based on carrying out the operation (or action) in a step until such time as the state of the transition changes. This makes the SFC language particularly suitable for programming sequential operations.

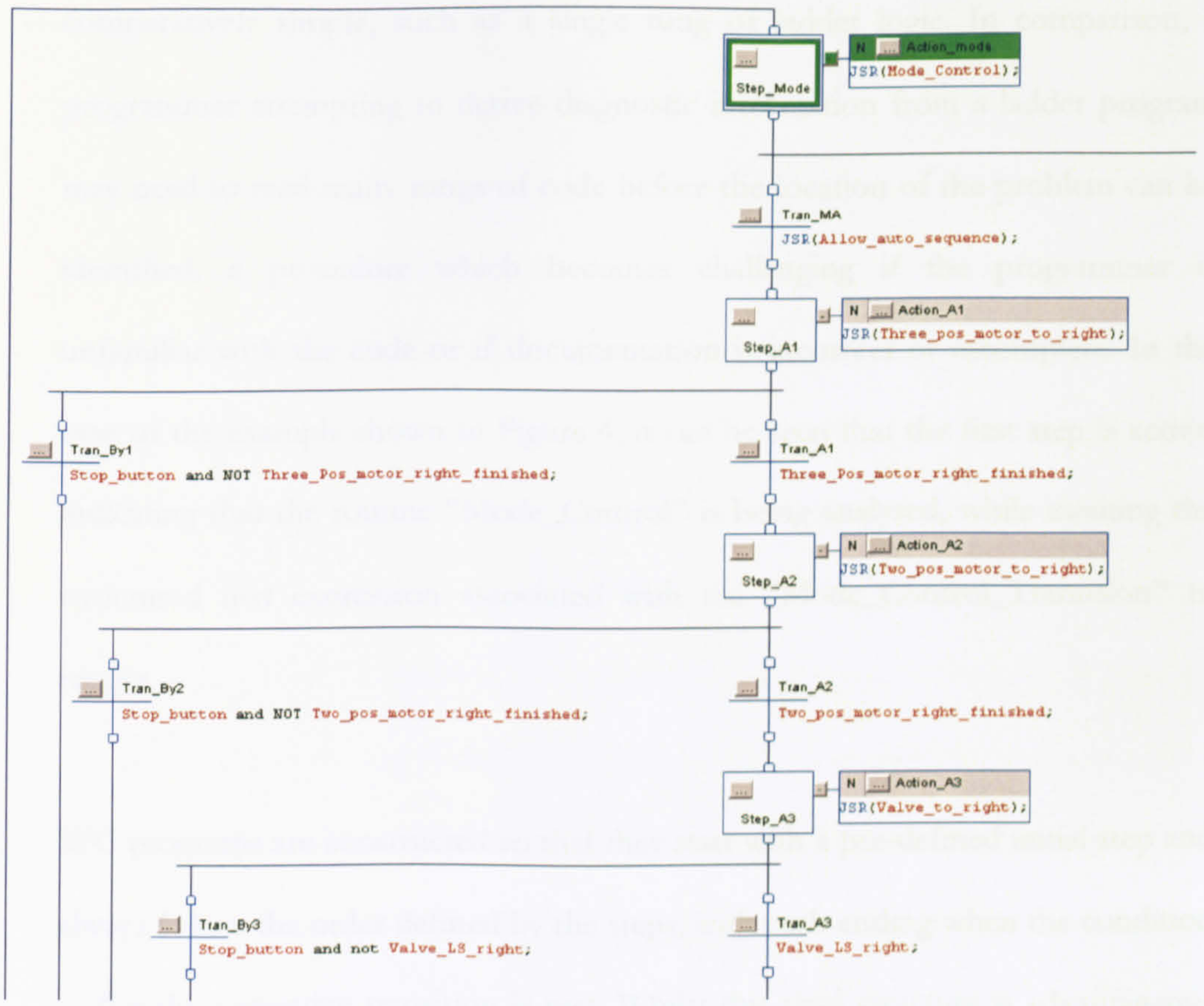

Figure 4 Example of Sequential Function Chart

In Figure 4, it can be seen that the first step is highlighted, indicating that the program is awaiting logic in the transition below the step to become true. In this case, a maintenance technician can immediately deduce that the system is waiting for a signal from the tags representing one of three buttons. The main advantage of an SFC is that it allows visualisation of the main states in a system together with all possible changes in state and the reasons why these changes could occur (Lewis, 1998). This also serves as a very powerful diagnostic tool: if a step is observed to be highlighted longer than anticipated, a maintenance technician looking at the SFC can deduce that the transition immediately following that step is waiting to fire. From this, attention can be focussed on the subroutine associated with that particular transition, a section of code which will generally be

comparatively simple, such as a single rung of ladder logic. In comparison, a programmer attempting to derive diagnostic information from a ladder program may need to read many rungs of code before the location of the problem can be identified, a procedure which becomes challenging if the programmer is unfamiliar with the code or if documentation is incorrect or incomplete. In the case of the example shown in Figure 4, it can be seen that the first step is active, indicating that the routine "Mode_Control" is being analysed, while awaiting the structured text expression associated with the "Mode_Control_Transition" to trigger.

SFC programs are constructed so that they start with a pre-defined initial step and always follow the order defined by the steps, with each ending when the condition to fire the respective transition is met. Whilst this rigid structure is advantageous for the creation of the desired sequence of operations, it can cause difficulties in terms of error recovery should the sequence not run as planned. Without additional work, the only mechanism by which a manufacturing sequence can recover from a fault is through the triggering of each transition in turn. The design of parallel branches to permit alternative paths through the process can help address this problem, though in doing so the complexity of the final solution is increased. In this sense, the flexibility of a ladder program may be better as the looser structure increases the ease with which error recovery functions are programmed.

2.1.2 Formalised Models

In view of the recognised limitations of ladder logic, it has been recommended that software design techniques used in commercial software engineering should be applied to address the problems faced in industrial applications (Edan and Pliskin, 2001). An example of this is the use of formalised programming tools in PLC's which draw from mainstream computer science. Work on formalisation has been conducted in a number of areas. One example is provided by (Young *et al*, 2000) who outline a method for decomposing a manufacturing cell into its constituent components, which in turn are modelled using UML and mapped to PLC code. Similarly, (Bani Younis and Frey, 2004) describe a method of converting PLC programs into platform independent XML models. Other ideas include the use of Finite State Machines, as applied by (Shah *et al*, 2002). The tool which is most used however is the Petri-net, an analytical tool created originally for the study of automata and Finite State Machines. According to (Rosell, 2004) the strength of Petri-nets lies in the fact that they present a unified modelling tool, providing a common approach to modelling systems, and include dynamic and adaptive behaviour suitable for application in areas such as assembly and task planning. At the same time, active control of systems using Petri-nets can be achieved by assigning inputs and outputs to the places and transitions of each net. There are however, recognised difficulties in terms of translating Petri-net models into executable code (Zurawski and Zhou, 1994). (Taholakian and Hales, 1997) achieved this through a model for mapping Petri-net models to ladder logic. Similarly, (Frey, 2000) notes a one-to-one correspondence between Petri-nets and commands written in an instruction list. At the same time, it is interesting to note that virtually all applications make use of ladder or structured text rather than SFC

even though (David, 1995) recognises that there are few differences between Petri-nets and Grafcet, a predecessor of SFC. The exception is work by (Carpanzano *et al*, 2004) in which a Petri-net model is realised using SFC.

2.1.3 Enterprise Controls

Although none of the formalised concepts described in 2.1.2 have been developed into commercial programming tools, Rockwell Automation has created programming tools based on object-modelling concepts for use in specific projects. These products allow programmers to define the functionality of a particular device within a software profile after which a code generation process is used to create the software to be used within the control application. Two types of object-modelling software, both known as Enterprise Controls (EC) were tested for use in controlling the facility shown in Figure 1, one producing compiled software and the other working on an interpreted principle. This testing provided first hand experience of the differences in usability between tools and helped inspire the research described in this innovation report.

Support for the compiled version of EC was withdrawn in early 2003, a decision partly influenced by the difficulties encountered in the implementation conducted here. The interpreted version of EC remains available on the market.

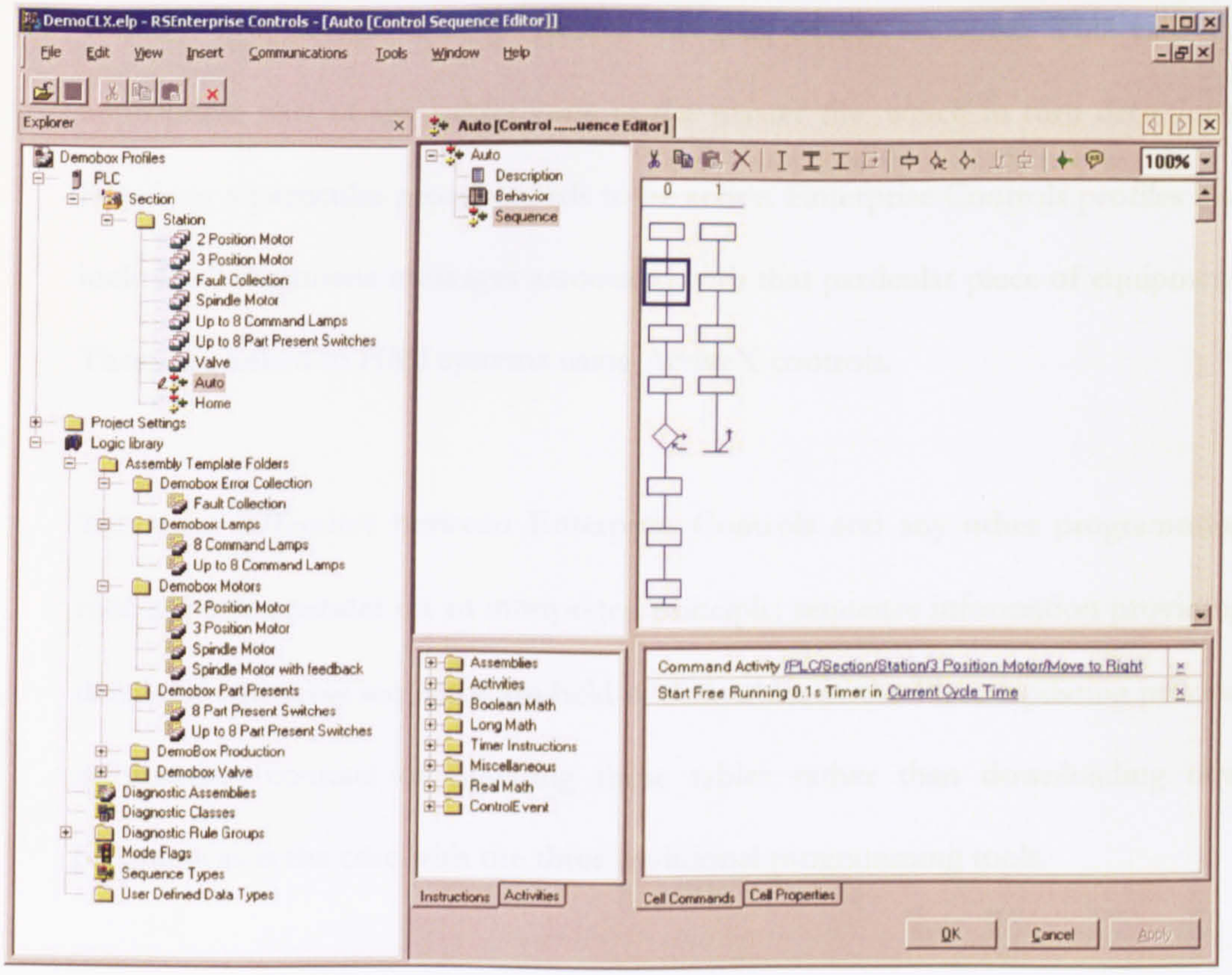

Figure 5 Enterprise Controls

The interpreted version of Enterprise Controls is based on the creation of device (or assembly) profiles which contain the functionality necessary to operate a specific piece of equipment. The Enterprise Controls programming tool (shown in Figure 5) works with a master ladder logic file which contains the functionality to operate machinery. This ladder logic file makes use of a traditional step-based structure. Once profile templates have been created, instances can be defined in which the signals defined in the profiles are associated with real hardware tags, as defined in the master ladder logic file. An automatic generation process is then used to create a new ladder logic file specific for the particular situation. Enterprise Controls also has a sequence editor, which looks similar to an SFC in that it consists of rectangular cells, similar to steps used in SFC programming. Cells can be assembled to form a sequential operation. Each rectangular cell

contains a command appropriate for that part of the sequence. This calls the appropriate part of the ladder code in the master file, which in turn determines how long a particular process needs to be active. Enterprise Controls profiles also include all diagnostic messages associated with that particular piece of equipment. These are linked to HMI systems using ActiveX controls.

The main difference between Enterprise Controls and any other programming tool is that it operates on an interpreted principle: sequence information providing details of a process sequence are held in data tables in the PLC. Updating process information consists of updating these tables rather than downloading new programs as is the case with the three traditional programming tools.

2.2 INDUSTRIAL PRACTICE

A first insight into how logic control programs are written in industry is provided by (Lucas and Tilbury, 2003a) who report on an observational study of the software design process. Within this paper, it is reported that the reluctance of industry to adopt alternative programming methodologies owes much to the fact that the benefits of a switch have not yet been demonstrated. The authors propose an assessment method based on the construction of a fully featured development environment but reject this on the grounds of excessive cost. The primary conclusion of this work, which is the only known report detailing how industrial software is written, is that the logic design process is heavily reliant on experienced programmers who adapt existing code to suit the particular application in question. Reference is made to specification documents but the details contained within this documentation are not provided.

2.3 COMPARISONS

Although a direct comparison between the IEC61131 languages has not been conducted, prior research has looked at comparisons between formalised ideas and ladder logic. A good starting point is comparison work which has been conducted by (Venkatesh *et al*, 1994). In this paper, programs written in ladder logic were compared with Petri-nets. This was achieved through conducting an analysis of the number of elements used in each programming environment, concluding that Petri-nets are a more effective programming tool. (Lee and Hsu, 2004) recognise the limitations of this work, and conduct a further analysis, in this instance analysing the number of logical expressions in comparable programs, again concluding that Petri-net models are better. It is however noted in both cases that Petri-nets are difficult to realise in practice. (Taholakian and Hales, 1997) address this through their methodology for developing ladder logic expressions from Petri-net constructs. A common feature within this work is that it relies on the conversion of Petri-net programs into PLC code. Although in some cases the structure of the ladder code developed from Petri-net models is described, the architecture of the original ladder program providing the original benchmark is not outlined in detail. As well as the difficulty in adapting Petri-net models to ladder code, a further limitation of the measures used here is the applicability of the basic element or logical construct comparison method to other formalised or object-oriented programming tools.

(Lucas and Tilbury, 2002) build on this earlier work and conduct a study based on comparing the use of different programming approaches to operate a reconfigurable manufacturing line. One of the measures they investigate is the

amount of time taken to create a program using tools such as Petri-nets and Finite State Machines. The investigation omits ladder logic as their original program was produced professionally, and is based on a sample of one individual working with each tool in turn. Process modification is mentioned in passing but not in any great detail. (Hajarnavis and Young, 2005b) provides a similar comparison between SFC and the commercial object modelling programming tool 'Enterprise Controls'.

2.4 CONCLUSIONS AND IMPLICATIONS

The main contribution of submission 2 is the identification of a gap in knowledge in terms of strengths and weaknesses of the programming languages and concepts available for programming PLC's: ladder logic, instruction list, structured text, function block, SFC and a variety of ideas for formalised programming tools. It is also noted that the context in which each of these languages is used by industrial users is not known and that although comparisons between ladder logic and formalised programming tools have been conducted in the past, the comparisons have all been found to be limited in scope and the methodologies used for achieving them are not necessarily suitable for wider application. Furthermore, all of these comparisons base their work on the ladder logic concept without describing the specific software structure under test. The results are also not expressed in a form in which they can be understood by industrial practitioners and are therefore unlikely to be understood and accepted, and by implication applied for use in real projects. There is therefore scope for addressing these limitations, firstly through the identification of code structures in use at present

and subsequently through conducting an evaluation which addresses the limitations of existing work.

3.0 INVESTIGATION INTO CONTROL SYSTEM DESIGN TECHNIQUES IN THE AUTOMOTIVE INDUSTRY

Having identified that previous research conducted to date lacks industrial relevance, the first challenge which has to be met is the capture and documentation of how end users approach the development and maintenance of their factory control systems, as well as how the limitations presented by use of ladder logic are overcome. It was thought that this could be best accomplished through contact with customers of the supporting company, and submission 3 provides full details of how capture of this information was achieved.

Inspiration for this work followed from a seminar at the University of Warwick in which an object-modelling programming tool (EC) was presented to a group of control engineers from the automotive industry. Following this seminar, the delegates were presented with a questionnaire in order to obtain some preliminary information about their practice in plant. This in turn inspired the idea of conducting a more detailed investigation into the use of control systems in industry. This chapter reports on the main findings of this investigation. Full details are provided in submission 3.

3.1 INVESTIGATION METHODOLOGY

The mechanism chosen for this work was a series of semi-structured interviews with a number of control systems planners in the car industry, the majority of which were conducted face to face. A further interview was conducted using a combination of e-mail and telephone call to expand on the initial information provided. The alternative idea of distributing postal or e-mail questionnaires was

eliminated owing to the small sample and consequent impact following an expected low completion rate, as is often seen with this type of research. Each interview was based on a set of 40 questions covering areas including PLC hardware choice, software structures in use, network and communication technology as well as addressing mechanisms for selection of products and desirable goals, features and characteristics for the future. These questions are listed in submission 3. The methodology selected here was designed to be open so as to enable capture of information about an area for which there is no prior knowledge.

The industry segment chosen for this investigation was the car industry, concentrating specifically on body assembly. This choice was taken owing to collaboration with the UK Automotive team at Rockwell Automation, with body assembly chosen owing to the high level of automation used in this area. A key feature and advantage of this work was that it provided exposure to customers of the primary competitor to Rockwell Automation, Siemens, thus providing some information regarding alternative systems, albeit not as detailed as that which could be obtained from the supporting company.

3.2 PARTICIPATING COMPANIES

A range of companies were approached for their assistance in this investigation including all of the volume car manufacturers with a presence in the UK (at the time of investigation, this consisted of BMW, Honda, Jaguar Land Rover, MG Rover, Nissan, Peugeot, Toyota and Vauxhall (General Motors). Given that many of these companies were either unable to take part, or could not provide relevant

information, the investigation was extended to Germany, where Audi, BMW, DaimlerChrysler, Ford and VW agreed to provide access to individuals to be interviewed. A subsequent approach was made to Ford, DaimlerChrysler and General Motors in the United States in order to obtain further information to supplement that obtained in Europe. DaimlerChrysler agreed to contribute towards this study.

Full details relating to each company are provided in submission 3. In the interests of preserving company confidentiality, company names and references to practice followed by specific organisations have been omitted from the results presented in this innovation report.

3.3 SUMMARY OF FINDINGS

A summary of the main findings of this investigation are shown in Figure 6 which lists the factors identified within each interview, identified by line numbers for reference in the main text. From this, the similarities and differences between the respective companies can be seen. Overall, this table provides useful indicative information about the nature of current industrial practice together with some of the problems and challenges faced by users of control systems in the car industry. This report deals with factors of most relevance to the main discussion and analysis of other areas identified in the survey are included in submission 3.

The main similarity between the participating manufacturers is the use of company-specific standards for ensuring consistency, increasing system transparency, and in many cases, global standardisation. The main difference seen

was the method by which this was realised, with participants selecting standards using software languages and structures taking one of three forms: contact based ladder logic, in which operations are controlled by machine conditions (Figure 2), step-based ladder logic (Figure 3), in which operations are controlled by a variable step number and a combination of the IEC61131 languages with sequential operations structured with SFC. Very little of this information is in the public domain at present, with the best example being a book published in 2003 which introduces the concept of the Ford EDDI standard (Parr, 2003). This information on EDDI is however, very limited in nature. Although use of the step ladder concept has been available for use by programmers since numerical evaluation capabilities were implemented within processors, EDDI is thought to be the first company standard to make use of this idea.

The disadvantage following from the use of company standards is that the adoption of alternative methods to those defined in the standard is discouraged. This is reflected in the fact that familiarity is identified as playing a part in the selection of a PLC and the adoption of a particular technique. Cost of training personnel is another reason for reluctance to migrate to different standards. It can therefore be argued that standardisation has an adverse effect in terms of the development of ideas and mindsets leading to the rejection of new ideas because they differ from practice defined for use in that particular company standard.

As can be seen in Figure 6 (line 66), also noted by many interviewees was the desire to conduct fast, correct and error free process changes in short production windows, with these most likely to be required during the start-up phase of an assembly line, cycle time improvements when in production, implementation of

changes to product specification and the introduction of new models and variants onto an existing line. In some companies, the implementation of process changes was thought to be problematic and error prone, whereas others were happy that their system architectures allowed the ability to cope with modification. One key concern expressed by many companies was ensuring that diagnostic information was kept synchronised with the control function after process changes had been completed.

All of the companies were seen to use some sort of Human Machine Interface (HMI) system to provide access to diagnostic information. Despite investing in this type of system, the fact that standards place emphasis on the ability to view and understand control code suggests a lack of faith in their fault visualisation systems, although only one respondent was willing to state this directly, as can be seen in Figure 6 (line 73).

				Company 1	Company 2	Company 3	Company 4	Company 5	Company 6	Company 7	Company 8	Company 9
PLC Choice	1	Omron	1							X		
	2	Rockwell	2		X			X	X			
	3	Schneider	3				X				X	
	4	Siemens	4	X	X	X	X			X	X	X
	5	Yaskawa (re-badged Modicon)	5							X		
Basis for Choice	6	Familiarity & previous knowledge	6	X	X	X					X	X
	7	Global or divisional standardisation	7		X	X		X				
	8	Purchase cost	8			X	X				X	
	9	Reliability	9	X								
	10	Impact on users	10				X					
	11	Support package	11	X	X	X	X		X		X	
	12	Training	12			X						
	13	Technical (e g language set)	13	X	X	X	X		X			X
Support for Soft PLC	14	Positive	14								X	
	15	Neutral	15	X						X		
	16	Negative	16		X		X					X
Important PLC / Control System Features	17	Programming language	17		X				X	X		
	18	Cycle times / speed	18	X			X					
	19	High speed communications	19						X		X	
	20	Expandability	20				X				X	
	21	Accurate process control	21	X								
	22	Ease of configuration	22								X	
	23	Support for large amounts of I/O	23								X	
	24	Good diagnostics & visualisation	24		X	X					X	X
	25	Dynamic diagnostics	25						X			
	26	Minimisation of physical connections	26						X			
	27	Minimise need to read code	27		X							
	28	Component interchangeability	28									X
	29	Communications & connectivity to other systems	29	X	X		X		X			
	30	Function libraries	30									X
	31	Local support	31		X							
	32	User friendly programming interface / ease of programming	32								X	X
	33	Reliability	33	X	X		X					X
Industrial Networks	34	ControlNet	34		X				X			
	35	DeviceNet	35									
	36	Ethernet	36		X	X			X			X
	37	Profibus	37		X							X
	38	Interbus	38		X	X			X			X
	39	Serial - RS232 / 485	39		X							
	40	Safetybus	40		X	X						
Use of Standards	41		41	X	X	X	X	X	X	X	X	X
Standard Aims	42	Global system commonality / increase system transparency	42	X	X	X			X		X	
	43	Bringing tested ideas into production environment	43									X
	44	Reduction in software development & commissioning time	44		X							
	45	Code / HMI consistency	45				X	X			X	
	46	Reduction in spare part stock	46	X		X						
	47	Cost reduction - training, bulk discount for purchases	47		X	X						X
	48	Personnel mobility	48			X			X		X	X
	49	Ease of initial algorithm implementation	49		X							
	50	Ease of in-house sequence change implementation	50		X							
	51	Automatic generation of diagnostic messaging	51		X				X			
Programming Language Choice	52	Ladder - Steps	52		X				X			
	53	Ladder - Contact	53					X		X		X
	54	Function Block Structures	54	X	X	X	X					X
	55	SFC / Mixed	55	X	X	X	X			X		
Compliance check	56	Manual - by managers, operators and maintenance personnel	56		X	X		X	X	X	X	
	57	Automated tools (fully or partly)	57				X					
	58	Not considered necessary	58									X
Main problems & challenges	59	Inadequate error descriptions	59		X							
	60	Lack of adherance to standard	60					X	X		X	
	61	Building trust in HMI systems	61		X						X	
	62	Gaining familiarity with new standard	62		X							
	63	Maintaining version control between online and offline files during development	63									X
	64	Configuration of fieldbus networks	64								X	
	65	Re-ordering sequences in ladder logic	65								X	
	66	Implementation of fast, correct, error-free process changes (in short production windows)	66	X	X			X	X		X	
Need for code viewing or modification	67	Missing / insufficient diagnostics	67		X	X						
	68	Cycle time or quality improvements, introduction of additional diagnostic messaging	68			X	X			X		
	69	Addition of features	69								X	
	70	Incorrect or incomplete operations	70								X	
	71	Fault condition not considered in standard / complex fault	71	X	X							X
	72	Language / translation problem	72		X							
	73	Mistrust in HMI	73		X							
Process flexibility definition	74	Ability to change model type seamlessly	74		X							
	75	Ability to make small changes quickly	75									X
	76	Ability to change the order of a set of operations	76								X	
	77	Building different parts with common infrastructure	77				X					
	78	Building several models on a common platform	78	X								
	79	Ability to build new body types, including one-off	79		X							
Desirable goals	80	Single network (including safety)	80	X	X	X	X		X		X	
	81	Open networks	81				X					
	82	Definition of industry-wide standard interface mechanisms	82	X								
	83	Breaking of link between hardware and software platforms / cross platform software portability	83			X					X	
	84	Faster introduction of new models	84									X
	85	Improved diagnostic capabilites in standard	85		X							

Figure 6 Summary of Interview Findings

3.4 CONCLUSIONS

The main contribution of this work is the identification that end users in the car industry make use of standards to define the software structures to be used in plant in order to aid understandability and improve consistency of software used in plant control systems. These standards define specific software architectures to be used within the respective programs. When conducting analysis of control software, it is therefore too simplistic to consider "ladder logic" as a language or concept alone and it is necessary to consider program structure alongside the language itself.

Familiarity with existing systems and the cost of training were identified as factors influencing the selection of a particular PLC and software architecture, suggesting that the human factors as well as technical and cost considerations play a part in the evaluation of tools for use in a project. At the same time, the fact that "familiarity" is specified as a factor considered affecting choice of PLC suggests that in order to encourage change in working practice, substantial benefits need to be demonstrated in order for them to be attractive to end users.

The importance of accurate diagnostic information was noted by many contributors as can be seen from Figure 6 (line 24). Their lack of faith in HMI systems suggests that design of control systems should be addressed through a systems-based approach in which control and diagnostic function are created and developed together rather than looking solely at whether a language alone can provide sufficient diagnostic information.

The open methodology used in this study was necessary in order to gain awareness of much of the information reported in submission 3. This provides the basis for a repeat of this evaluation in a more formalised manner (for example, asking users to rank each factor in order of importance) and could in turn direct research, development and marketing effort in the future. Prior to this investigation, these factors had not been recorded and therefore these results provide a strong base for future investigation in this area.

4.0 PROPOSAL FOR COMPARING PLC SOFTWARE DESIGN METHODOLOGIES

Building on preliminary work, this chapter describes an experimental plan for evaluating the productivity benefits provided by certain logic design methods – contact ladder logic, step-based ladder logic, SFC and a commercial formalised programming tool called Enterprise Controls (EC). The proposal, initially presented in submission 4 and subsequently modified slightly (as reported in submission 5) provides a mechanism for assessing the strengths and weaknesses of the logic design methodologies under test. This chapter gives an overview of the key features of the experiment plan as conducted, combining descriptive sections of both submissions 4 and 5.

4.1 SELECTION OF MEASUREMENT PARAMETERS

The first step in conducting experimental work is the definition of the output variables. Here, direct measurement of time and effort were proposed as appropriate measures to be used in this instance. Justification for this choice of parameters is provided here.

4.1.1 Time

It has been noted by (Das, 1996) that machine flexibility can be assessed by taking into account the efficiency of a machine – or in this context the amount of time that a changeover from one configuration to another takes with respect to the time a machine is in production. It follows that a machine with a short changeover time will be available for use more often than one in which this is a

lengthy process, and can thus be seen as being more flexible. The concept of flexibility is described as inherently vague by (Tsourveloudis and Phillis, 1998), who state that it is very much dependent on human perception. They identify a number of parameters which have an impact on machine flexibility: setup or changeover time, versatility and adjustability. Setup or changeover time is stated to be made up of time to prepare and reposition tools, and a negligible software changeover time. Software configuration time may be an insignificant factor in certain contexts, such as parameter setting in machine tools but in other environments, software modification time may be a major part of the equipment configuration process and it seems overly simplistic to neglect software modification time in its entirety. Although this work suggests that software changeover time is not a significant factor at present, it may become a challenge in the future as the development of flexible jigs and fixtures will require a corresponding improvement in software development time to enable best use of these new tools and techniques. The other parameters in the paper - versatility and adjustability - are more relevant in mechanical contexts than software, which by its very nature is highly versatile and adjustable.

From this, we can identify that time is a good comparison parameter for evaluating logic design methodologies. Unlike concepts such as software complexity, time has the advantage of being generic in nature, and thus suitable for use when comparing a range of diverse programming approaches which could not be compared using alternative means. From the perspective of an end-user of an automation system, time also has an advantage in that it can be expressed as a cost, both in terms of the cost of lost production but also as the engineering cost associated with the implementation of a changeover. The results can therefore be

expressed in simple terms to which industrial users can relate. This has a further benefit in that it can be used to help justify the value of one methodology or system over another, and thus helps address earlier claims that the benefits of one approach over another have not been fully demonstrated.

4.1.2 Effort

Time alone, however, does not provide the full perspective regarding a particular programming tool. One approach may require a short amount of time for the implementation of a series of complex commands, whereas another might require a large amount of time in which to accomplish a series of simple operations. A measurement of programming effort can therefore be used to supplement time data obtained from users performing a pre-determined task.

Given that most PLC programming packages run on PC's, a look at how programs are developed can provide a useful lead for measurement of programming effort. Most modern PC packages make use of a mouse and keyboard for input of user data. Capture of the number of operations implemented or steps taken can therefore provide an indication of the amount of effort required of a user. This assumes that an operation or technique which needs a large number of keystrokes or mouse clicks is indicative of more physical effort than one in which the same change in functionality can be achieved with fewer keystrokes or mouse clicks. One can conclude that an approach resulting in a lower key or mouse count requires less physical effort than one requiring a large key count.

(Lucas and Tilbury, 2003b) recognise the value of using task analysis for assessing ability to complete and evaluate a task, referring to work by (Card, 1980) and (Kieras, 1988, 1997) to give an overview of how this has been achieved in the past. In the case of (Card, 1980), the emphasis is on the experimental determination of the time taken to perform a keystroke level operation and the use of this data to predict the time taken for an expert to complete a task. The work by (Kieras, 1988, 1997) looks at this from a more abstract level, looking at functions such as "add a module". (Lucas and Tilbury, 2003b) use this work as a preliminary predictor of performance.

There is therefore a precedent for the collection of user interface data – but unlike the work of (Lucas and Tilbury, 2003b) this will be used to supplement time measurements rather than to verify them. There is no realistic alternative to key strokes and mouse clicks for capturing information of this nature.

4.2 FACTORS OF INTEREST

Having identified appropriate measurement parameters, the next phase of experimental work design is the determination of appropriate parts of the PLC software creation and modification process which are of interest for investigation. In this instance, the primary factors of interest chosen were the level of training or experience needed for a programmer to use a programming tool correctly together with measurement of the time and effort needed in order to complete a set task correctly. This data was supplemented by information on the tool selected by participants as that found easiest to use. This choice was based on two observations – from literature which suggests that process flexibility has not been

evaluated in detail, and from the industrial survey which suggests lack of objectivity in their project evaluations.

4.2.1 Impact of Experience

The impact of experience can be assessed from two perspectives – firstly, in terms of whether the skill level of a participant has an effect on their ability to complete a set task correctly and secondly through evaluation of whether prior experience of a programming tool has an effect on the choice of tool found easiest to use. Evaluation of the impact of experience on performance can be achieved by conducting a straight comparison between the completion rates (and time and effort) measurements of trained and untrained programmers who are asked to conduct identical tests using the same tools. The impact of whether prior experience has an impact on tool choice can be achieved by comparing the preferences recorded by the untrained participants with those with professional experience of the tools. Both of these assessments are appropriate for use in this investigation.

The selection of participants from a wide range of companies, as well as participants with no prior experience of programming PLC's provided the means for accomplishing evaluation of the impact of experience.

4.2.2 Program Modification

The investigation methodology proposed in submission 4, can be applied to various aspects of PLC use, such as investigation of the level of training required

to use a tool, creation of control functionality or obtaining diagnostic information from a system in use. The factor of interest here is the ability to modify a program to take into account engineering changes necessary to realise cycle time improvements or product type or volume changes. Justification for this choice draws from two sources. Firstly, despite evidence that there are differences between programming tools in terms of process flexibility (Hajarnavis and Young, 2005b), detailed evaluation of the ability to modify control code has not been conducted within the research community. Secondly, the investigation conducted among automotive users of PLC's indicates that flexibility is not necessarily a factor considered when selecting a software structure for use in a project. Consequently, it was thought that this would make a suitable area for investigation.

This evaluation was conducted through assessment of whether a process change is conducted "right first time" together with measurements of time and effort needed in order to achieve correct completion of the task. This is necessary to give the results validity as time and effort values alone, without consideration of the outcome will yield meaningless results.

4.3 EXPERIMENT PROPOSAL

4.3.1 Participants

An assessment of the level of skill required to complete a task with each tool was achieved through selecting participants with different levels of experience. The main differentiator between the two groups was whether or not they had any prior

experience of programming PLC systems. The reasoning behind the inclusion of untrained participants was to gain an awareness of the effect of experience on performance in the experiment and to evaluate whether there were any differences between the tool selected as easiest to use between untrained and experienced participants.

The primary aim in selecting experienced participants was the involvement of individuals with prior knowledge of each of the tools under test. Capture of this information required the involvement of companies in Belgium, Germany, India, the United Kingdom and the United States, primarily among automotive end users and also among system integrators and the supporting company (an automation vendor) in order to capture as broad a range of experience as possible. After experimentation had commenced, it was found that the classification of all of these participants as experienced was a little too simplistic given the wide variety and depth of experiences and so the experienced group was subsequently reclassified according to their particular job function – as maintenance personnel, system planners, programmers and employees of Rockwell Automation.

The number of participants in each category was determined largely by the people the collaborating companies were willing to make available. Maintenance personnel were found to be the group to whom access was most difficult hence the number of people in this group is smaller than that in other categories. Further information about participants is provided in Table 1 in section 5.1.

4.3.2 Choice of Equipment

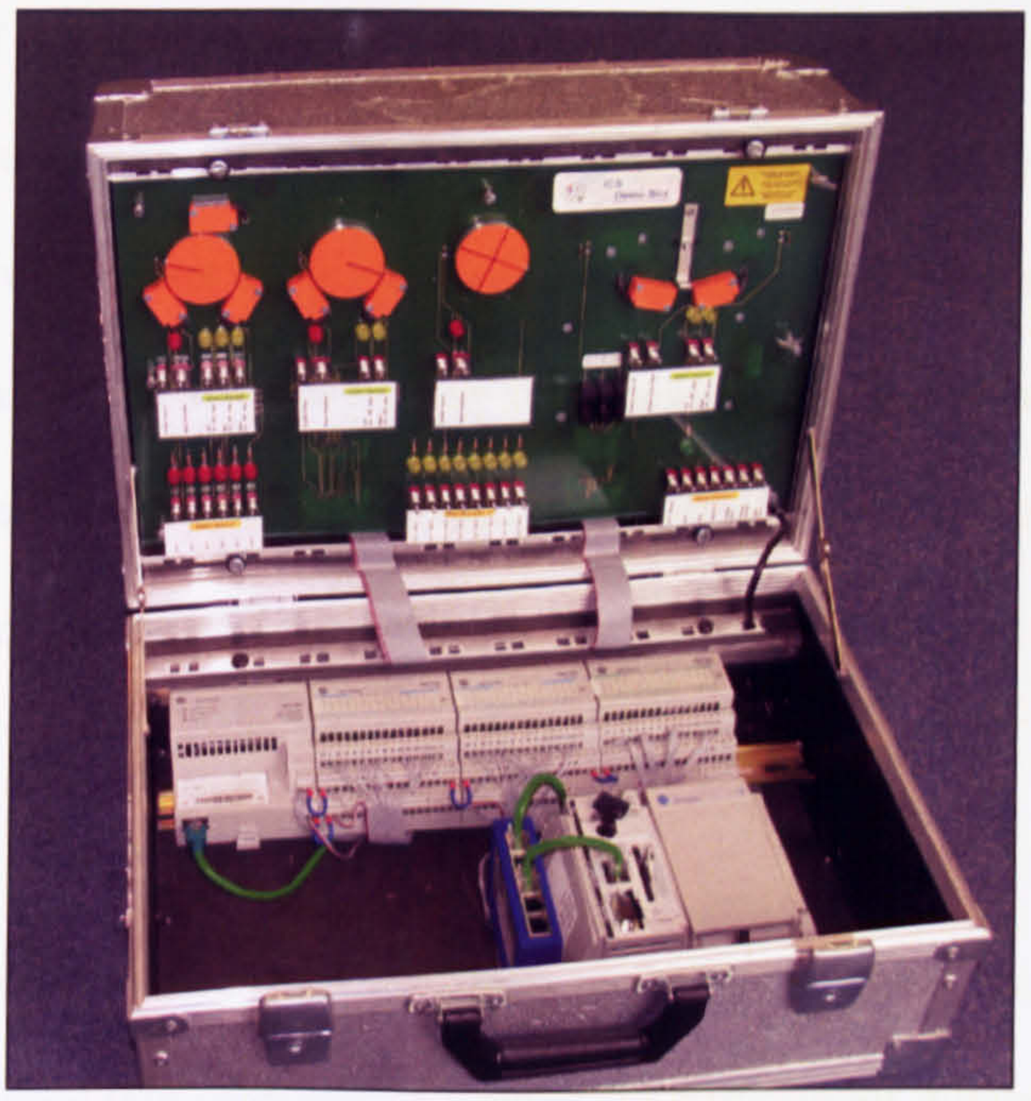

Figure 7 Demonstration Box

Experimental work required to achieve this comparison was conducted using a self-contained demonstration box containing a series of motors, lamps and switches controlled by a Rockwell CompactLogix PLC (Figure 7). The demonstration box consists of a set of devices designed to simulate typical features used within a manufacturing process and is made up of two bi-directional motors with limit switches to detect when each motor has reached the desired location, a unidirectional spindle motor with no positional feedback and a pair of relays designed to simulate a valve, again with limit sensors to indicate its position (left or right). The two motors are differentiated by the number of limit sensors fitted to them – one has three sensors allowing monitoring of whether the motor is in the left, middle or right positions. The second motor has two sensors, left and right.

4.3.3 Experimental Tasks

The basis of the comparison tests was a task set to each subject, who after a short amount of training and familiarisation was asked to modify a pre-determined sequence of operations using each of the programming concepts to be tested. The length of time taken to achieve this was measured, together with the number of keystrokes and mouse-clicks required in order to achieve the same functionality. Further evaluation considered whether the process change had been implemented correctly.

All participants were advised that they would be working to a 10 minute time limit. This served two purposes: it limited the experiment duration and also placed pressure on participants to complete the task quickly – effectively simulating a scenario in a factory in which production constraints require a task to be completed within a set time window. In practice, participants were allowed to overrun beyond this 10 minute limit if there was a reasonable chance that the participant would be able to present a solution. This was achieved by asking participants if more time was required.

Two main tasks were conducted within this exercise: a simple process modification and the identification of the failure causing a fault. A small number of willing participants were also asked to complete a more complex process modification. With the main process modification task, the expectation was for participants to change the functionality and deliver the new control function as well as updating messaging functions for correct interface to HMI and

appropriate adaptation of program comments so that the program retains clarity for future software changes.

In order to account for learning effects as the experimental tasks were completed, the order in which the participants were presented with each programming tool was changed from participant to participant. Practical considerations associated with the experimental method required minimal changeover of the HMI between experiments. In order to achieve this and at the same time to account for potential learning effects as the task was completed, if one individual completed the tasks first with EC, the following participant was asked to use EC last.

4.3.3.1. Task 1 – Changing Process

The task which participants were asked to conduct was the implementation of a simple process change. The initial sequence which was presented to all participants consisted of a cyclic set of operations. The aim of the task therefore was to swap two pairs of operations – the order in which the respective motors operate adapting the sequence shown in Figure 8 to match that in Figure 9. Although seemingly straightforward, it required care on the part of the programmer to avoid mixing conditions relating to inputs and outputs for each of the devices.

4.3.3.2. Task 2 – Fault Diagnosis

Although the comparative simplicity of the equipment used in the experiment prevented a full scale diagnostic test, it did allow the assessment of whether a

participant is able to detect the nature of a system fault generated without their knowledge. The idea here was to assess the participant's approach to the diagnostic task and to establish whether the first reaction was to make use of an error message displayed on the HMI or to analyse the software and therefore help establish whether continued access to diagnostic code is justified.

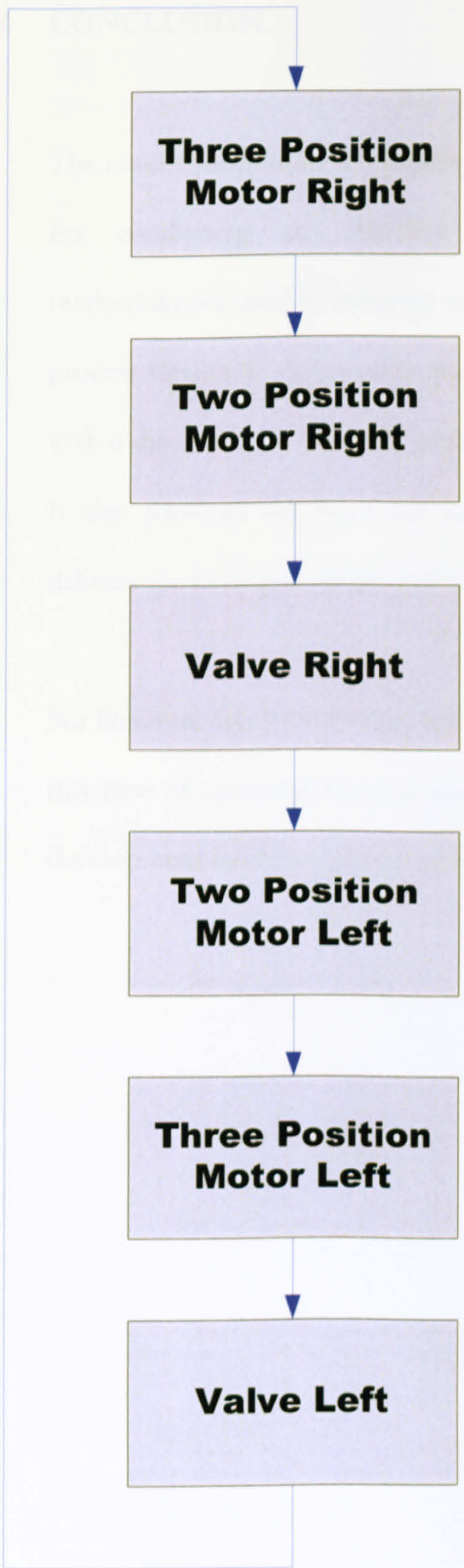

Figure 8 Original Sequence

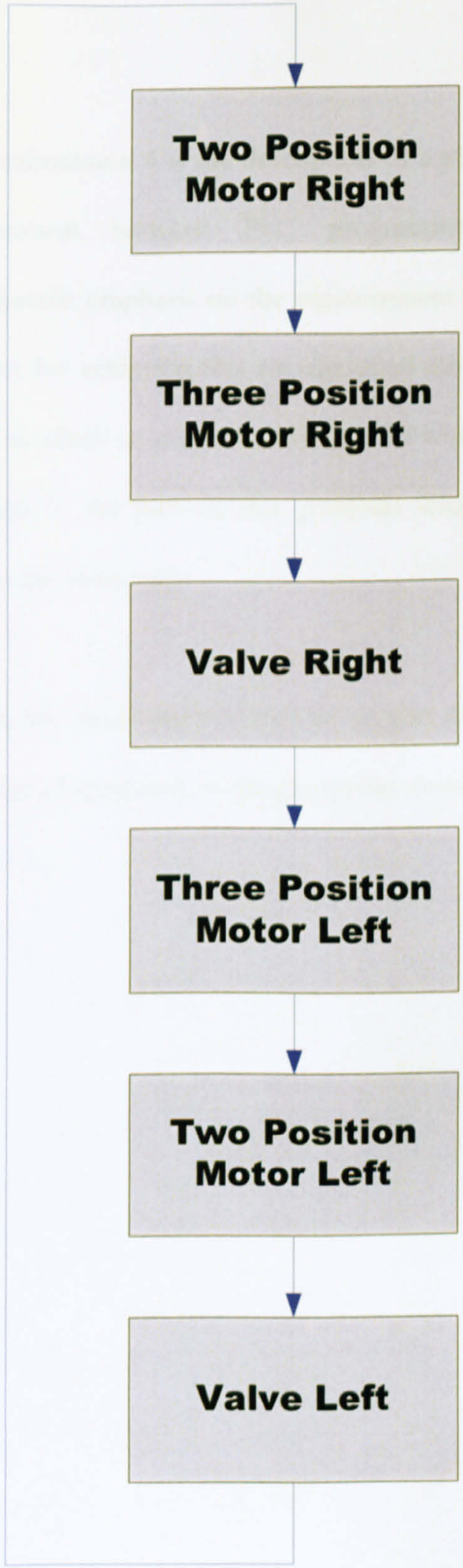

Figure 9 Modified Sequence

4.4 CONCLUSION

The main contribution of chapter 4 and submission 4 is the description of a plan for conducting an objective comparison between PLC programming methodologies used in industry, with a specific emphasis on the measurement of process flexibility. Appropriate parameters for achieving this are identified along with a discussion of potential platforms on which to conduct experimental work. It also provides the basis for submission 5, the part of this portfolio which delivers the main part of the innovation in this doctorate.

No literature has been found, nor is there any anecdotal evidence to suggest that this type of customer focused and task-based approach to programming system development has been attempted in the past.

5.0 ASSESSMENT OF PLC SOFTWARE STRUCTURE SUITABILITY FOR THE SUPPORT OF FLEXIBLE MANUFACTURING PROCESSES

Based on the results of the experimental work, and drawing mostly on material in submission 5, this chapter covers three main areas: identification of the programming tool which provides the best performance, an evaluation of the respective skill levels needed to achieve the task effectively with each programming tool and assessment of whether there is a difference in the tool of choice between trained and untrained participants in the experiment. Material presented here consists of results, analysis and implications, commencing with information about the individuals taking part in the investigation.

5.1 PARTICIPANTS

Category	Number of Participants	Average Self-Assessed Skill Level		
		Ladder	SFC	EC
Maintenance	6	3.83	1.83	0.33
Planners	15	4.00	1.53	1.40
Programmers	15	4.53	2.33	0.13
Rockwell	9	4.11	2.33	1.22
Untrained	18	1.17	0.33	0.33

Table 1 Participant Self-Assessed Skill Levels

Table 1 shows the number of participants in each category together with their average self-assessed skill level, where 0 indicates no prior knowledge and 5 shows highly proficient. Average skill level is also presented graphically in Figure 10, which also indicates error bars showing the maximum and minimum values in each category. It can be seen from the graph that in all categories, participants have greater knowledge of ladder logic than either of the other programming tools. A large range for non-ladder examples is the result of the way in which

participants have been classified and the fact that these tools are less established than ladder. For example, the definition for "untrained" participants is based on the idea that the individuals in this category are not professional PLC users. Therefore, a researcher at the University of Warwick, justifiably ranking his experience with EC as "5" was included in this group even though most people in this category had no prior knowledge and ranked themselves as "0". This gives a larger range of values than might otherwise be expected. With ladder, the range of values specified by maintenance and programmer categories is comparatively compact, reflecting the nature of the roles. In contrast, the range for planners is quite large – a reflection on the fact that participating planners had varying levels of hands-on programming experience.

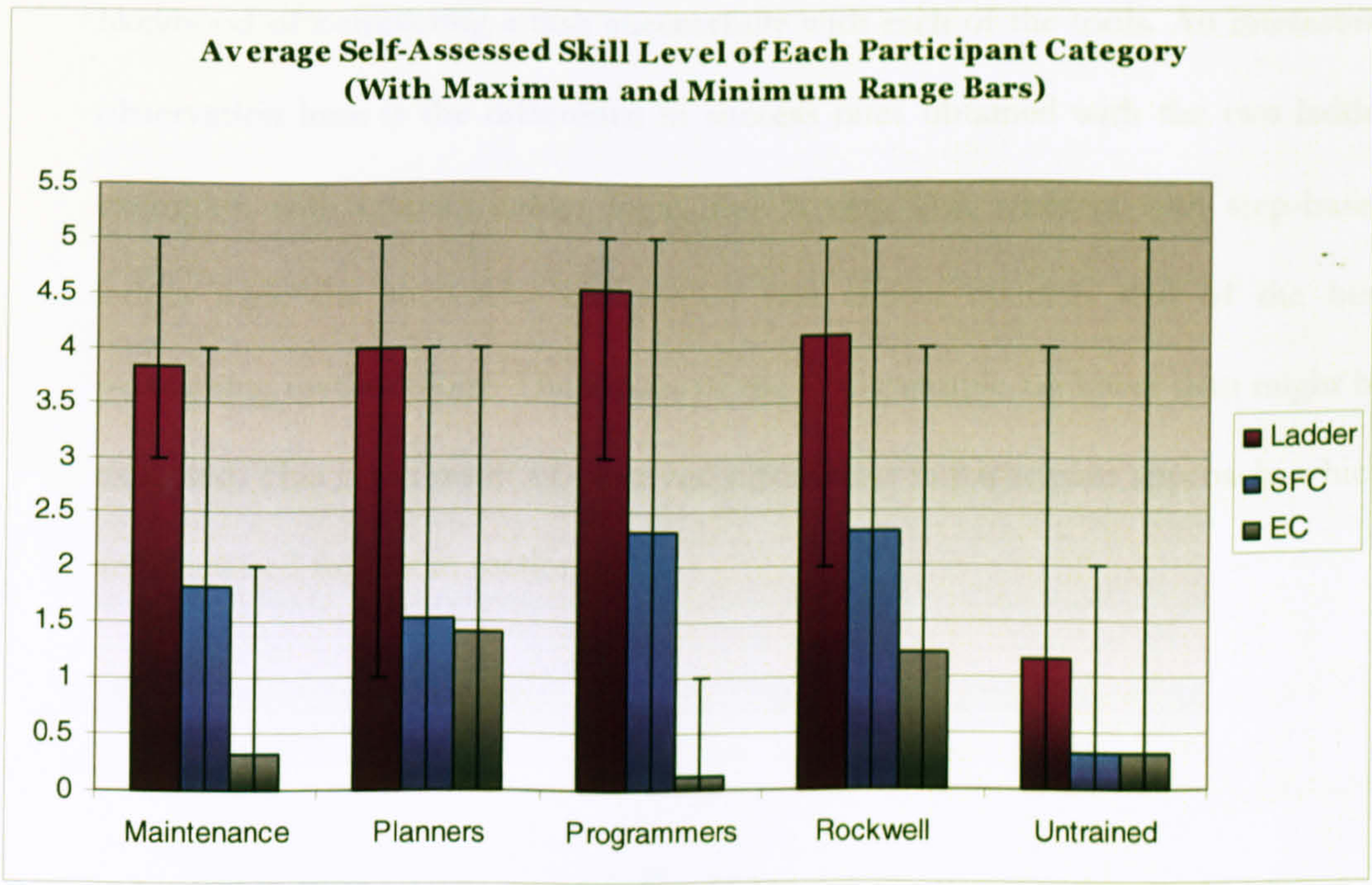

Figure 10 Average Self-Assessed Skill Level of Each Participant Category (With Maximum and Minimum Range Bars)

5.2 RESULTS AND ANALYSIS

5.2.1 Right first time data

Figure 11 shows the results of the "right first time" evaluation in graphical form. For each of the four programming methodologies, the successful completion rates are shown. The graph shows the overall aggregate figure for all participants in the exercise followed by a breakdown of each of the participant categories in turn. From the overall figures, it is clear that there is a considerable difference in performance between the four methodologies, with the lowest success rate (37%) achieved with contact ladder logic and the highest with EC (89%). Given that all four tasks were conducted by all participants, this gives an indication as to the likelihood of completing a task successfully with each of the tools. An interesting observation here is the difference in success rates obtained with the two ladder examples: with contact ladder logic, this is very low, whereas with step-based ladder logic the successful completion rate almost matches that of the best performing methodology. The results of the SFC example are lower than might be expected. This is the result of observed differences in participant approach, which are discussed further in section 5.2.2.

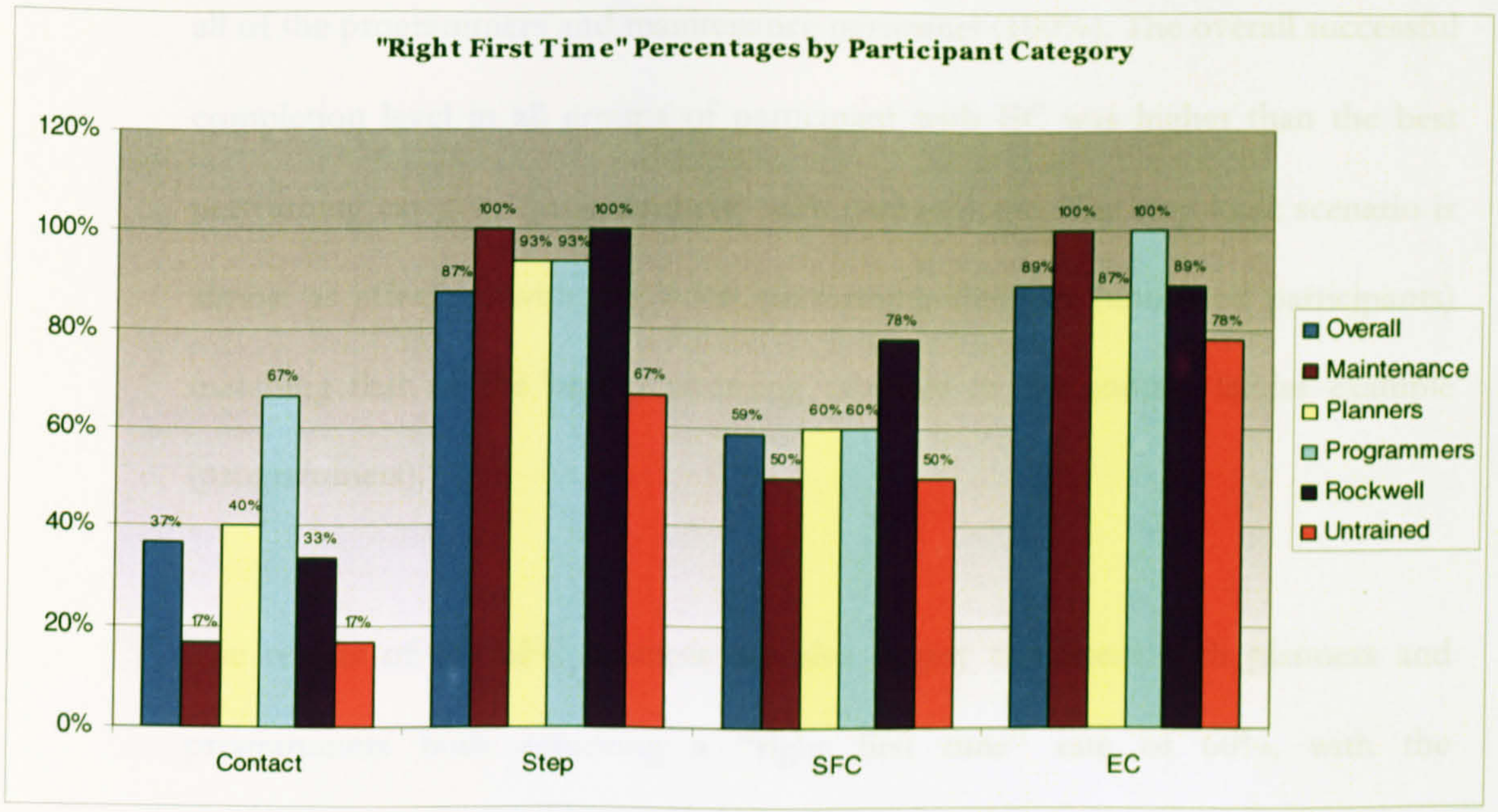

Figure 11 "Right First Time" Data Overall and by Category

It can also be seen in Figure 11 that there are differences between how individual categories of participant performed with each task. This difference is greatest in the contact ladder logic example in which it can be seen that of the 18 participants in the untrained category, only 3 (17%) were able to complete the task "right first time" compared to 10 of the 15 participants (67%) in the programmer category. This is not surprising given that programmers have greater familiarity with the ladder logic programming tool and are thus better placed to understand the complexity of a ladder program than their untrained counterparts. However, there are indications which suggest that this low completion rate is the result of the program structure rather than the language itself – the step ladder logic example demonstrates both a higher completion rate by all categories of participant, and smaller variation between the groups (the best is 100%, the worst 67%) unlike the contact logic scenario (where the best is 67% and the worst 17%). In contrast to the contact ladder scenario, the smallest level of variation is seen in the EC example in which 14 untrained participants (78%) completed successfully, as did

all of the programmers and maintenance personnel (100%). The overall successful completion level in all groups of participant with EC was higher than the best performing category (programmers) with contact logic. The step logic scenario is almost as effective, with the worst performing category (untrained participants) matching that of the best performing category in the contact ladder example (programmers).

The results of the SFC example are also largely consistent with planners and programmers both achieving a "right first time" rate of 60%, with the maintenance and untrained categories achieving a rate of 50%. The Rockwell category achieved a higher "right first time" rate of 78%. This is a reflection on the fact that the Rockwell SFC interface is not commonly used by industrial practitioners and can be confirmed by the self-assessed skill level scores shown in Table 1. The higher completion rate among the Rockwell employee category also supports the idea that the tool requires greater familiarity in order to be used effectively.

5.2.2 Differences in Approach

Observations conducted during the experimental work show that participants followed different approaches to conducting the task for some of the four methodologies. This was particularly noticeable in the step ladder logic and SFC scenarios, each of which presented a number of distinct approaches for completing the task. With the step logic example, most participants either chose to change the step numbers in the code (the expected solution) or they achieved the process change by changing tag allocations in the PLC code, with a handful of

participants opting to use a combination of the two approaches. Similarly, most participants working on the SFC example either opted to rearrange and reconnect the elements of the SFC or they modified the routine calls forming the process. The remainder again used a combination of the two approaches or attempted to achieve the change through modification of step tag names (e.g. renaming step 1 as step 2 etc), an operation which does not deliver the required change in functionality.

5.2.2.1. SFC

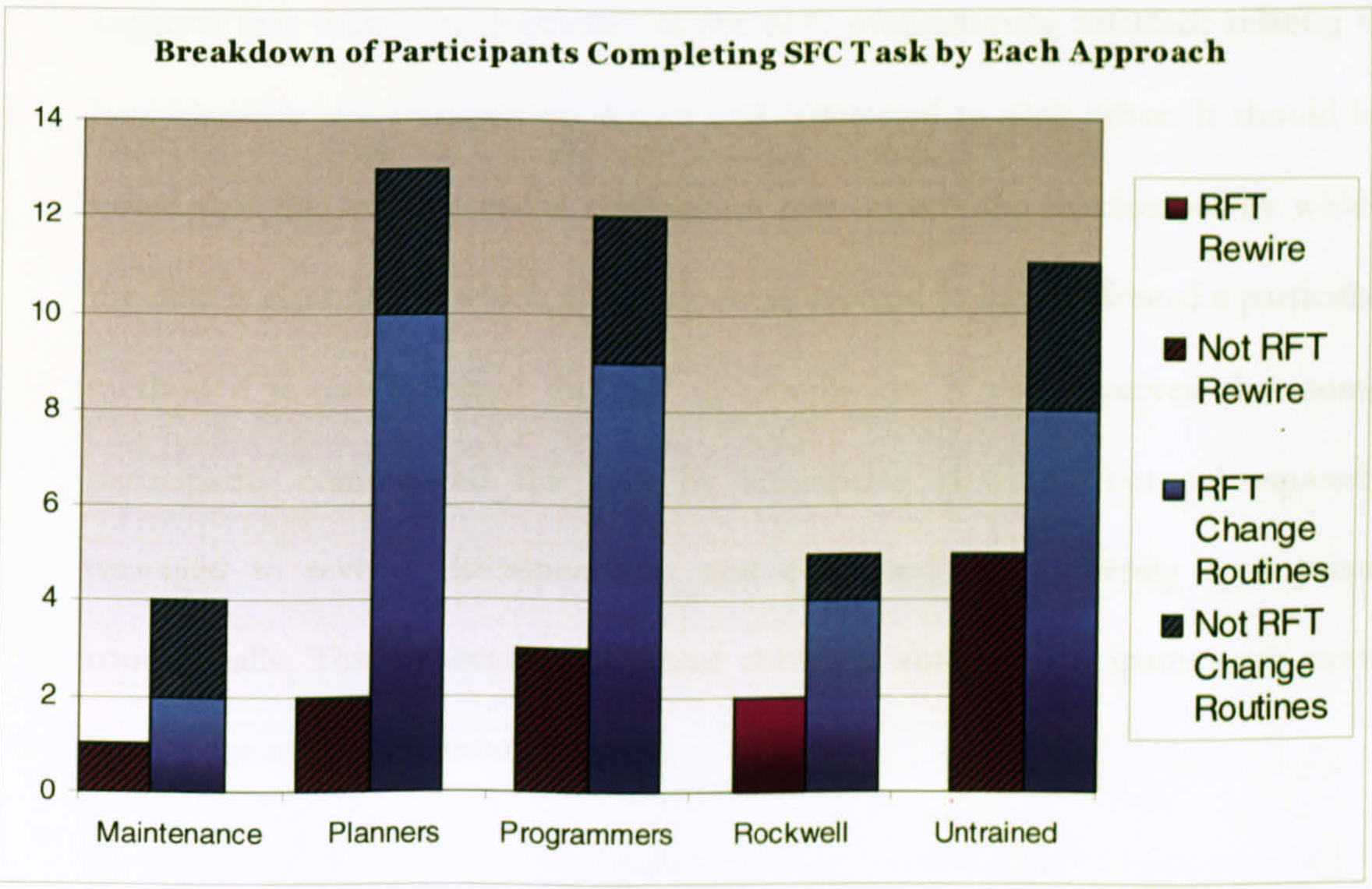

Figure 12 Number of Participants Completing SFC Task using Each Approach

Figure 12 shows the number of participants approaching the SFC example in each of the two ways, along with the "right first time" (RFT) completion rate. This shows that there is a clear difference in successful completion rate with a success rate consistently above 50% among participants who opted to change routine calls

compared to an almost universal failure rate among those who opted to rewire the SFC. The two participants who completed successfully by rearranging elements of the SFC had extensive knowledge of the programming interface (both were employees of Rockwell Automation, and one of these was the product manager for the RS Logix 5000 programming tool). Those who failed to complete generally ended up with a solution in which the program had not been changed at all or an SFC screen which did not compile owing to a wiring fault. In the case of participants who followed alternative approaches such as changing calls to ladder subroutines and structured text expressions in transitions, the reasons for failure were generally omissions and incorrect changes as well as syntax errors. This suggests that there is a deficiency in the SFC programming interface relating to how elements are arranged on screen and connected to each other. It should be noted that the low successful completion rate reflects the mechanism by which the data is classified in which a participant is deemed to have followed a particular method if it was followed through to completion: it was observed that some participants commenced the task by attempting re-wiring but subsequently managed to reverse the operations and continued by modifying appropriate routine calls. This behaviour was most common among participants with prior knowledge of the programming tools.

5.2.2.2. Step Logic

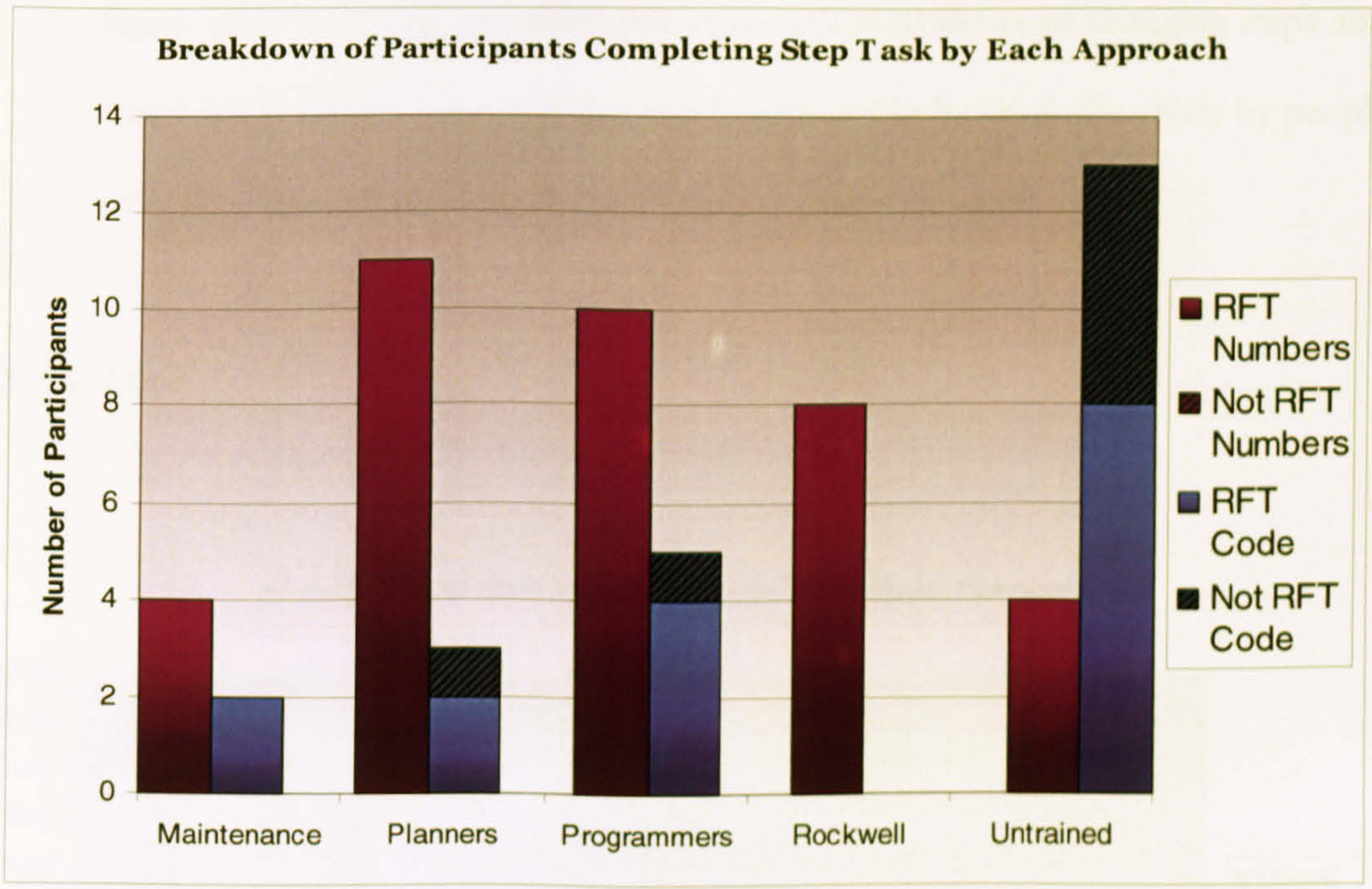

Figure 13 Number of Participants Completing Step Task using Each Approach

Figure 13 shows a breakdown of how each category of participant completed the step ladder logic task – by changing tag allocations in code or by changing step numbers – together with an indication of how many participants completed the task "right first time". In all categories other than the untrained group, the majority of participants opted to change the step numbers rather than changing tag allocations in the code. All participants who changed step numbers went on to complete the task correctly. In contrast, the majority of untrained participants opted to achieve the process change through modification of the control code, with a reasonably high success rate of 62%. One reason why untrained participants, with limited prior knowledge of PLC programming opted for this option is the limited information provided in the briefing sheets at the outset of the experiment. This would suggest that in order to use the technique most

effectively, additional training beyond the limited information provided in the briefing sheets is required. At the same time, the overall successful completion figure of 67% among untrained participants (a total figure of changing steps and code) is a good indicator that the step logic tool can be used effectively by people with very limited prior knowledge of how it should be used.

5.2.3 Time

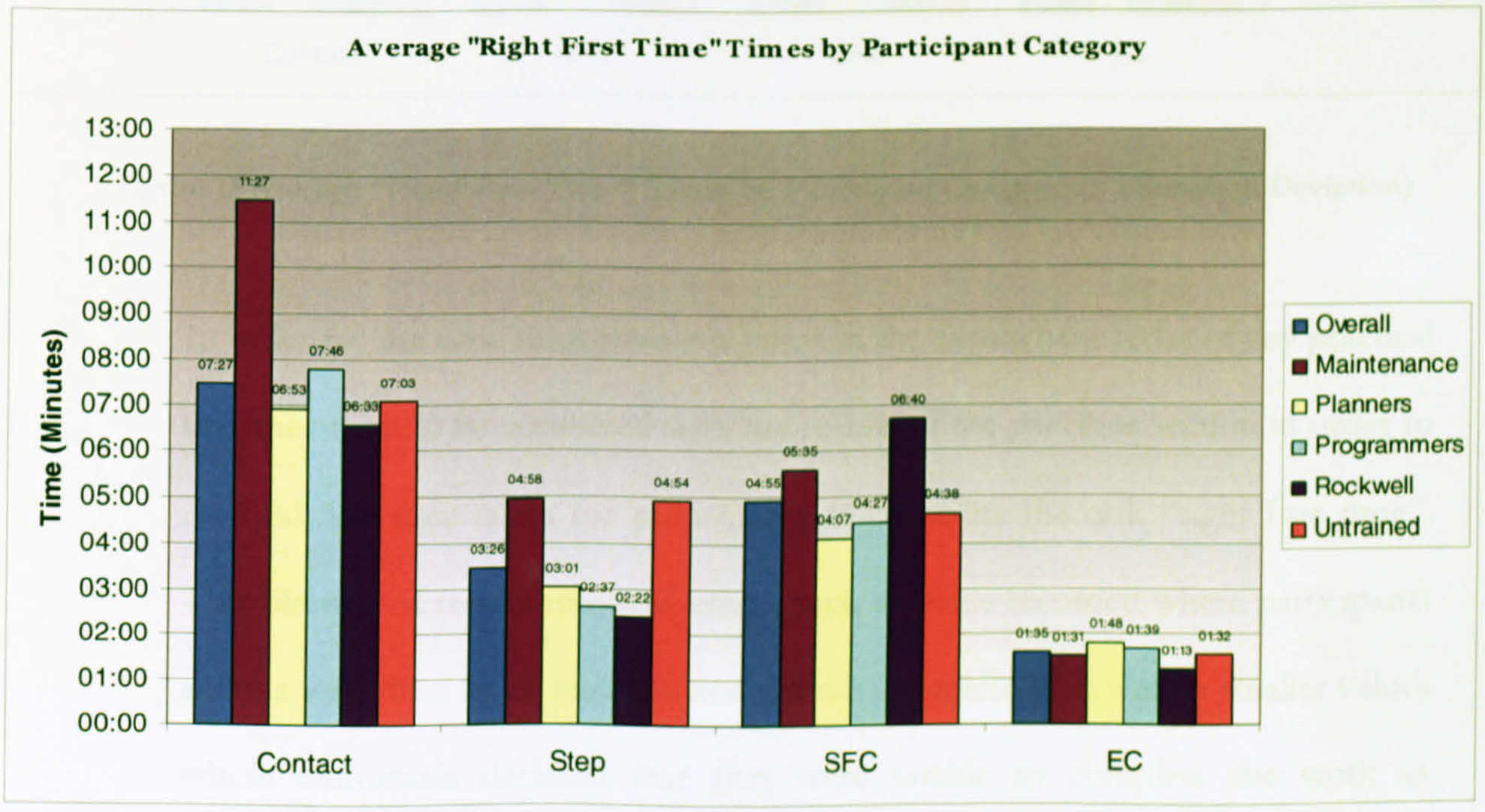

Figure 14 Average "Right First Time" Times by Participant Category

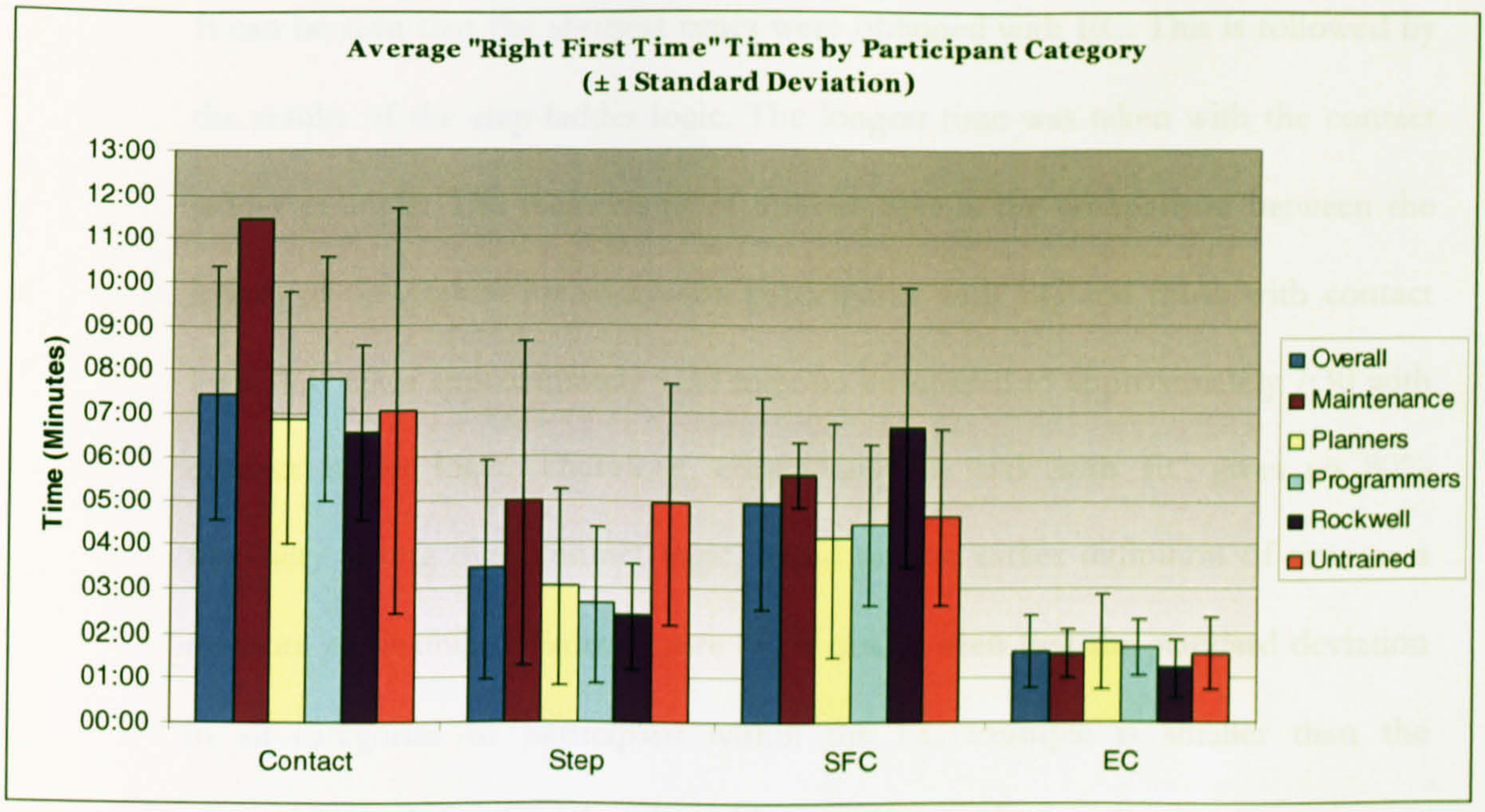

Figure 15 Average "Right First Time" Times by Participant Category (± 1 Standard Deviation)

In order for the time measurements taken in the experiment to be of any practical use, they need to be combined with the results of the previous section in order to establish the time taken for participants to complete the task "right first time". This filtering of results removes values such as those recorded where participants spent a long time on an example and did not complete it, as well as smaller values where individuals declared that they were unable to complete the work as required. Figure 14 shows the mean times recorded with each programming tool, initially showing an overall value for all 63 participants, and subsequently broken down into participant categories. Figure 15 presents the same data together with error bars indicating 1 standard deviation above and below the mean. No error bars are shown for the maintenance category completing the contact ladder example as the result is that for the single individual who completed the task and it is therefore impossible to calculate a standard deviation.

It can be seen that the shortest times were obtained with EC. This is followed by the results of the step ladder logic. The longest time was taken with the contact ladder example. The main result of interest here is the comparison between the length of time taken for successful participants with EC and those with contact logic. EC takes approximately 1:30 minutes compared to approximately 7:30 with contact ladder logic. Therefore, completing the task with EC gives an 80% flexibility saving over contact logic, based on the earlier definition of time as a measure of flexibility. From Figure 15, it can be seen that the standard deviation in all categories of participant within the EC example is smaller than the respective result for the other three programming tools.

Reasons for large standard deviation with step ladder and SFC are outlined in submission 5, which also provides details of the impact of participants choosing one method of completing the task over another with the SFC and step logic examples. Two main methods of completion were observed and the results are shown graphically in Figure 16 and Figure 17. These indicate that the average time spent when step numbers were changed was consistently lower than the situation in which tag addresses were changed in code (Figure 16 and Figure 17 display the respective mean values and error bars ± 1 standard deviation). The fastest changeover times are achieved by skilled participants modifying step numbers. Similarly, among participants completing the SFC example, most participants attempted to change routine calls or rearrange SFC elements. Separation of the results according to whether the task was completed through rearrangement of SFC elements or whether it was achieved through change of routine calls shows that the average time for participants completing the task through the modification of routine calls is seen to be considerably lower than those who

reorganised SFC elements. Overall, these results support the observations made regarding "right first time" completion and indicate that not only do certain methods of completing the task yield better completion rates but they are also complemented by shorter completion times.

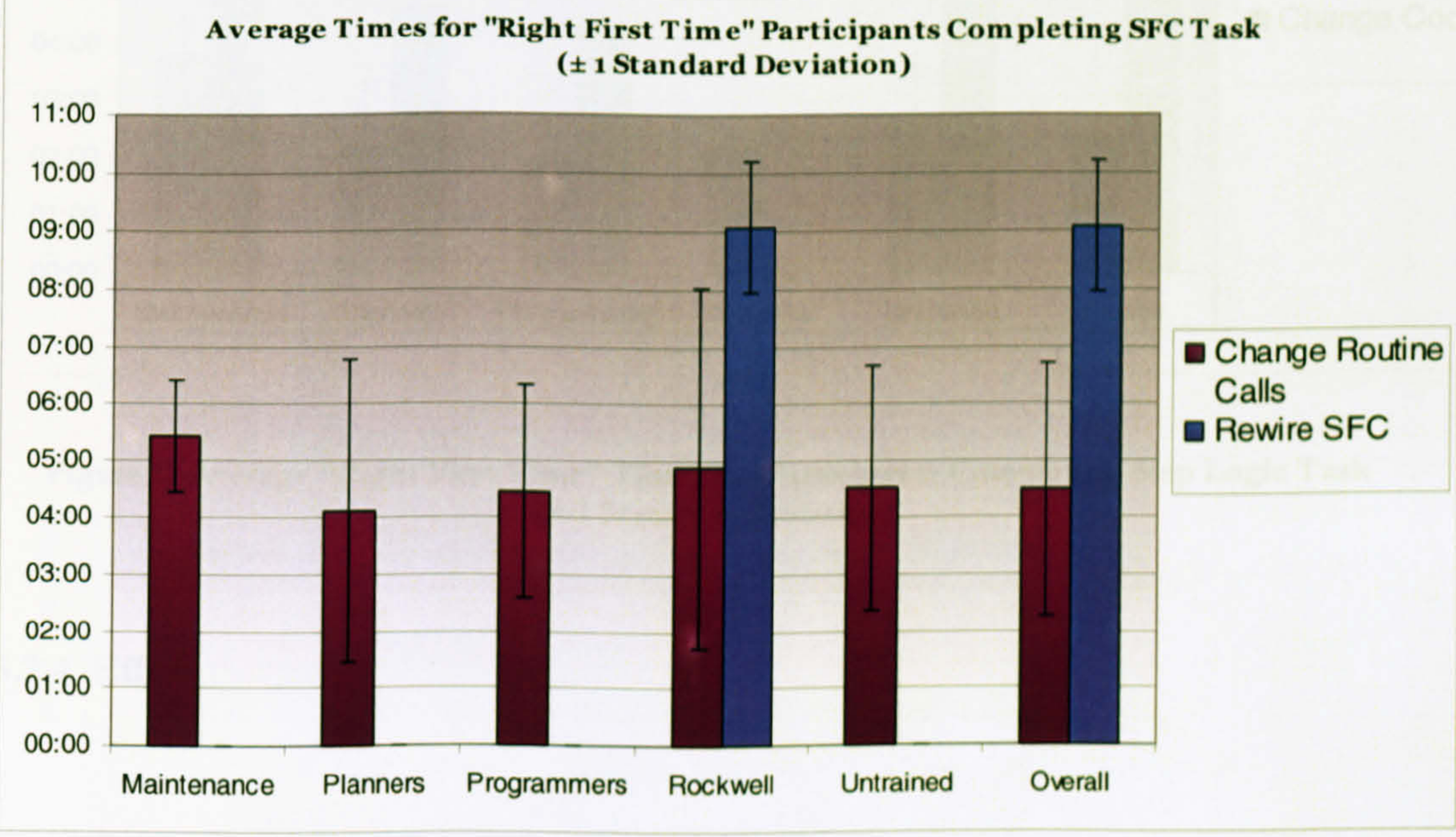

Figure 16 Average Times for "Right First Time" Participants Completing SFC Task (± 1 Standard Deviation)

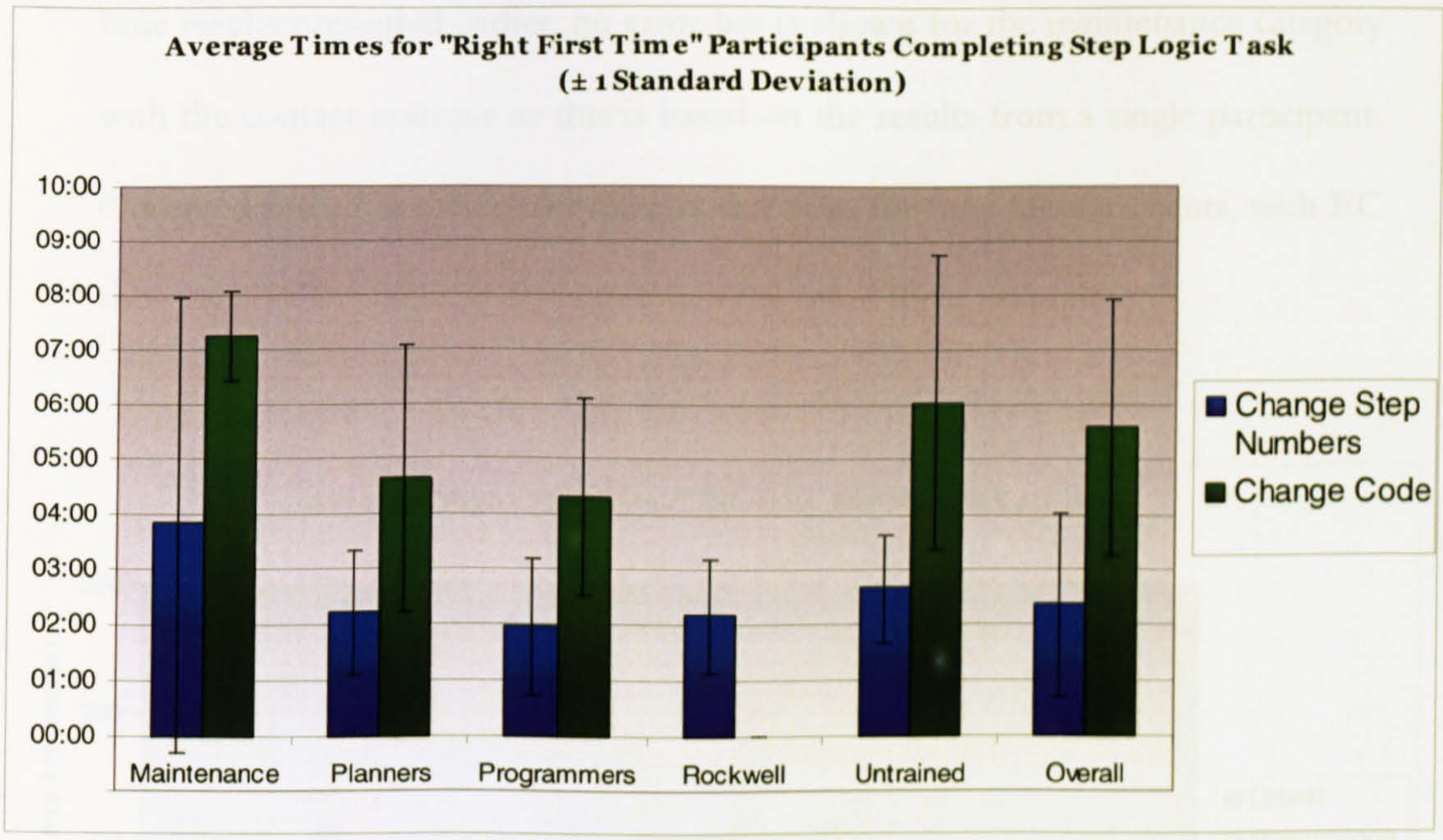

Figure 17 Average "Right First Time" Times for Participants Completing Step Logic Task (± 1 Standard Deviation)

5.2.4 Effort

Two separate parameters were measured to gain an indication of the physical effort made by the participants during the exercise: mouse clicks and key strokes – in effect recording the two mechanisms which the participants had in order to interface with the programming tool. They provide a useful indication as to how a task was completed. These results are presented as an aggregate of mouse clicks and key strokes for each example. Figure 18 shows the average effort score for those participants who completed the task "right first time". The results trend broadly reflects that for time spent on the task, with the contact logic example generally requiring the most amount of user input and EC the least. Unlike the time values however, there appears to be more cross category variation in effort. Figure 19 supplements the information in Figure 18 through the inclusion of error bars indicating ± 1 standard deviation around the mean. As is the case with the

time results presented earlier, no error bar is shown for the maintenance category with the contact example as this is based on the results from a single participant. Observed spread in the results mirrors that seen for time measurements, with EC showing smaller standard deviation than the other three examples.

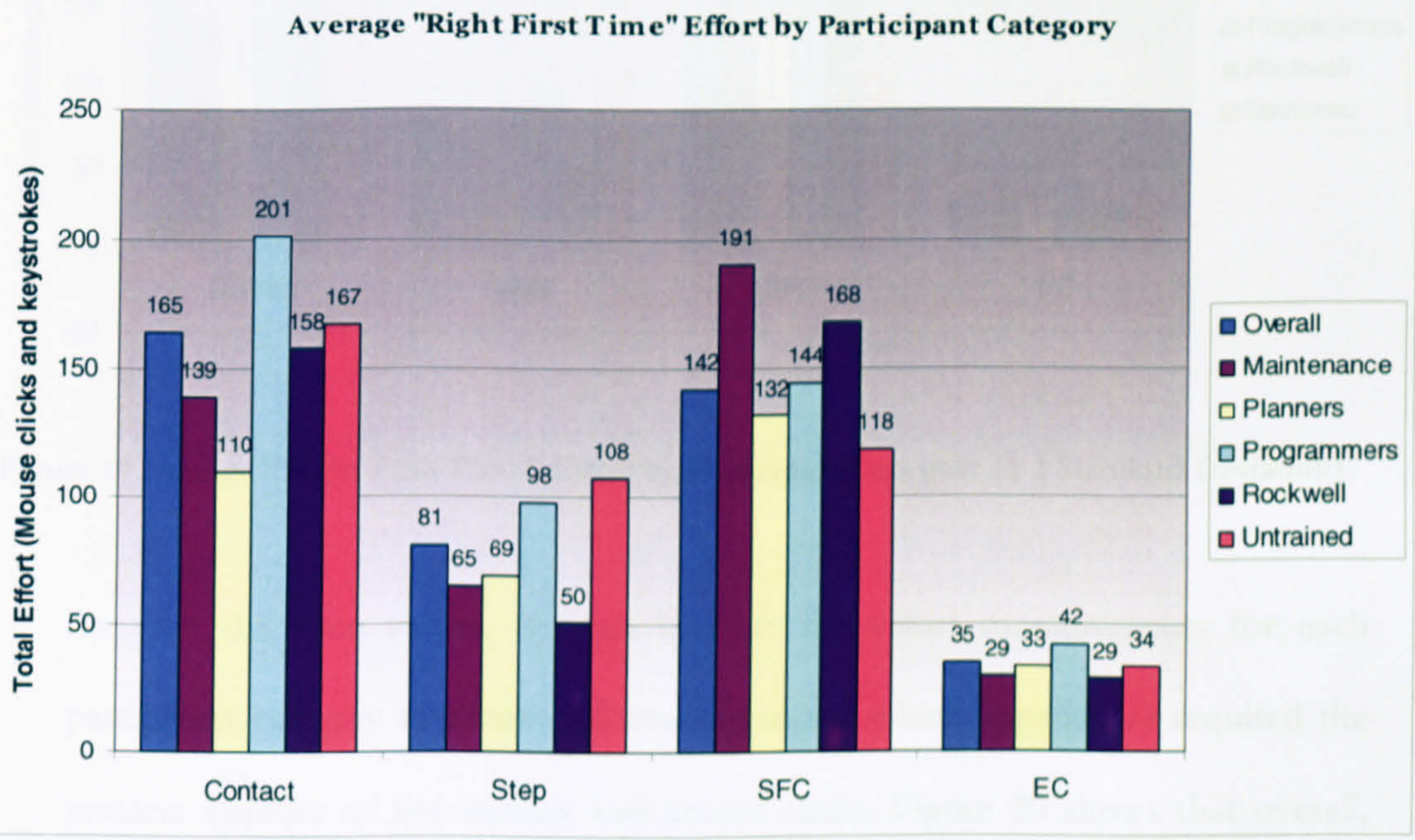

Figure 18 Average "Right First Time" Effort by Participant Category

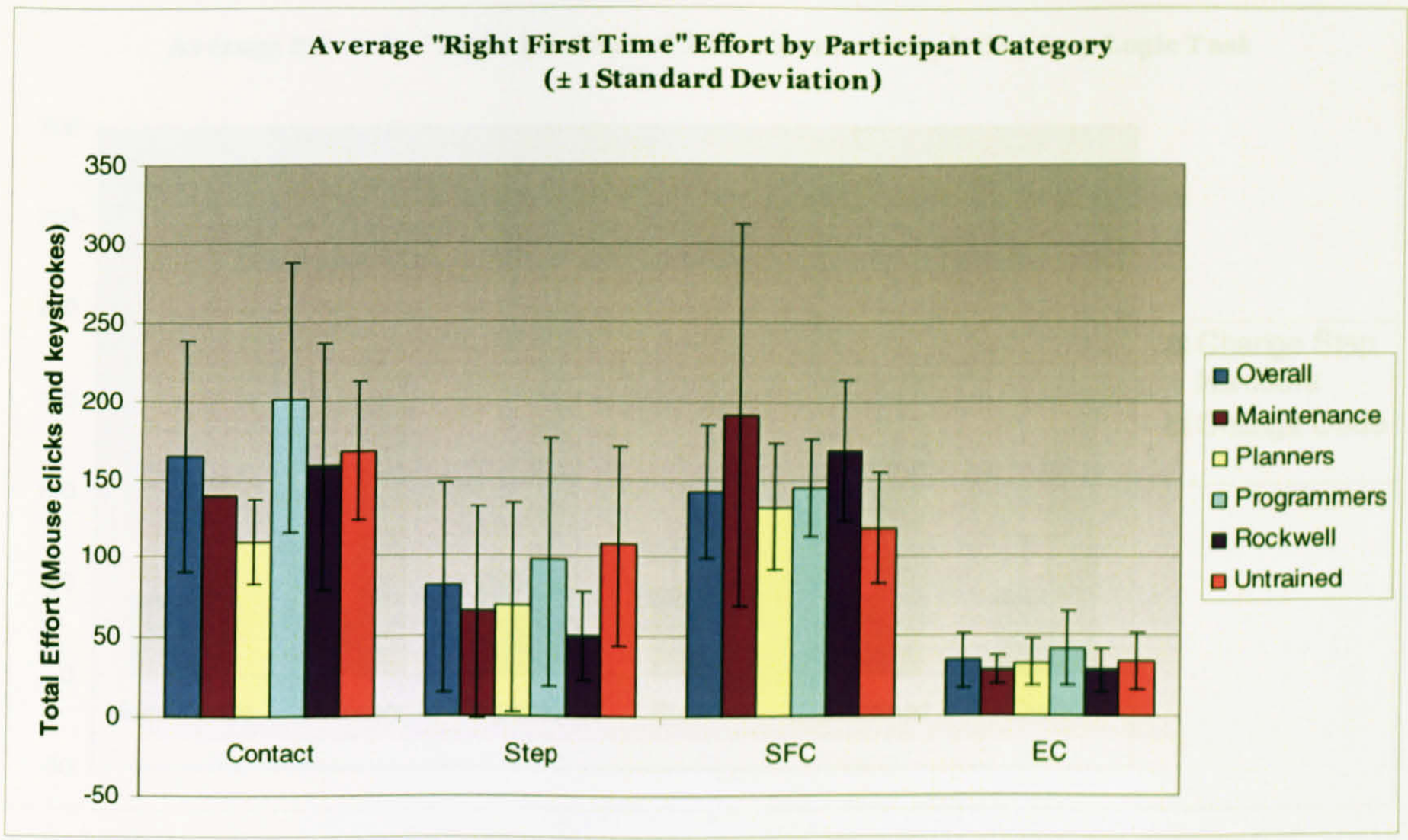

Figure 19 Average "Right First Time" Effort by Participant Category (± 1 Standard Deviation)

As with the time values, we can look at the effort measurements for each participant category to obtain information about which approaches required the greatest number of key strokes and mouse clicks. Figure 20 shows that overall, accomplishing the task by changing step numbers requires approximately a third of the effort needed to do so by working through the program and modifying each input and output in turn. Interestingly, the recorded values for untrained participants who modified code are lower than those for the programmers and planners. However, values for programmers and maintenance technicians are drawn from the small number of participants who completed the task using this approach so need to be treated with caution. There is more consistency within the participants who changed step numbers though again it appears that untrained participants provided less user input. In this instance, the untrained sample is small (4 individuals).

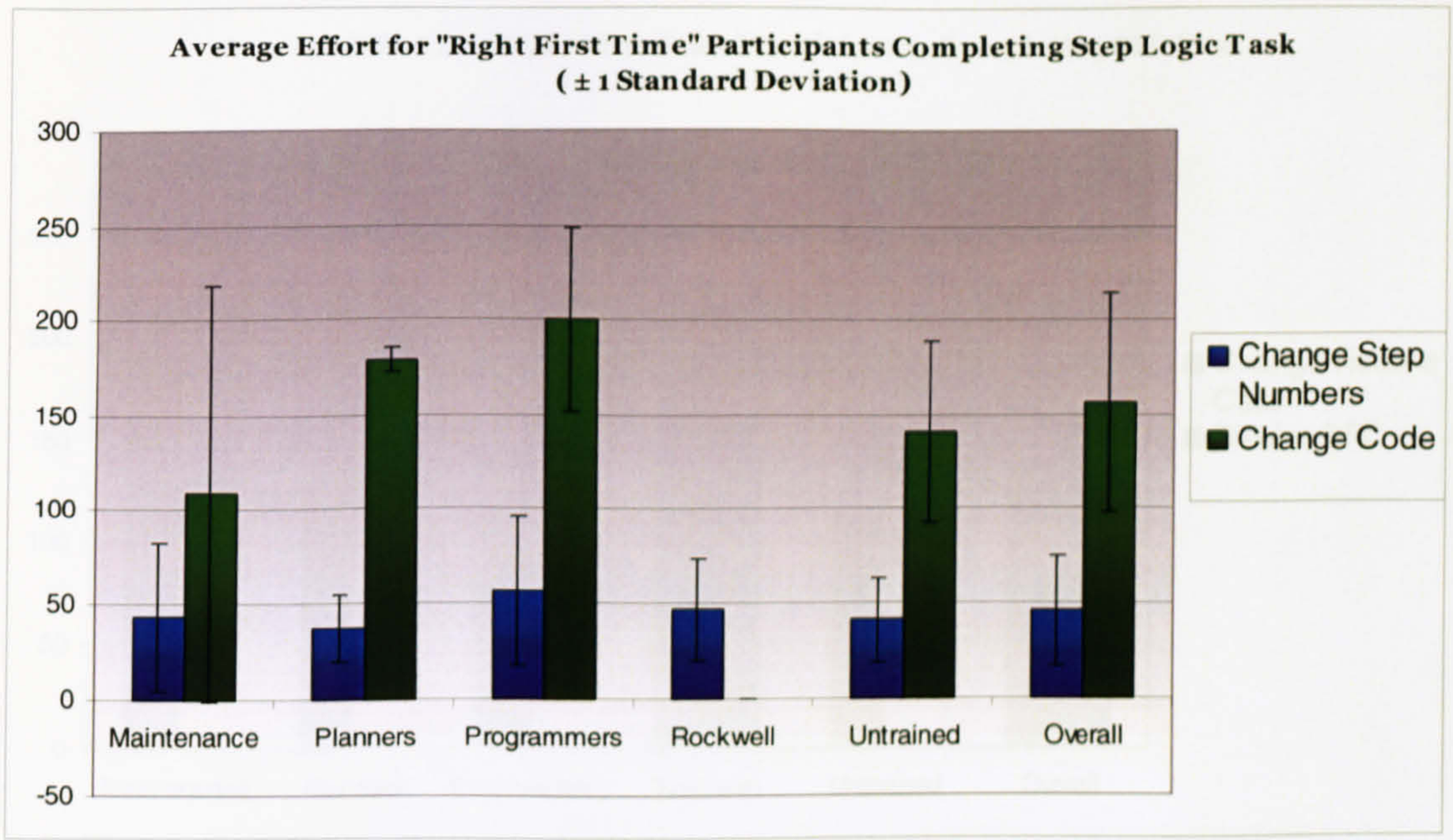

Figure 20 Average Effort for "Right First Time" Participants Completing Step Logic Task (± 1 Standard Deviation)

Figure 21 shows similar data for the two approaches seen by participants working with the SFC example. At first glance, it appears that to change the SFC elements by rewiring requires greater user input than to modify a set of routine calls though once again this is limited by the fact that this task was completed by just two participants.

Regarding the results for the data resulting from changes to routine calls, the eight untrained participants were seen to require less effort than any of the other types of participant. This again supports the idea that participants in the more skilled categories may have commenced using one approach but seen the task through to completion with the other – where the training of the participants in the maintenance, planner and Rockwell categories have the knowledge to undo operations and continue using another method, the untrained participants who encountered difficulties will not have completed the task.

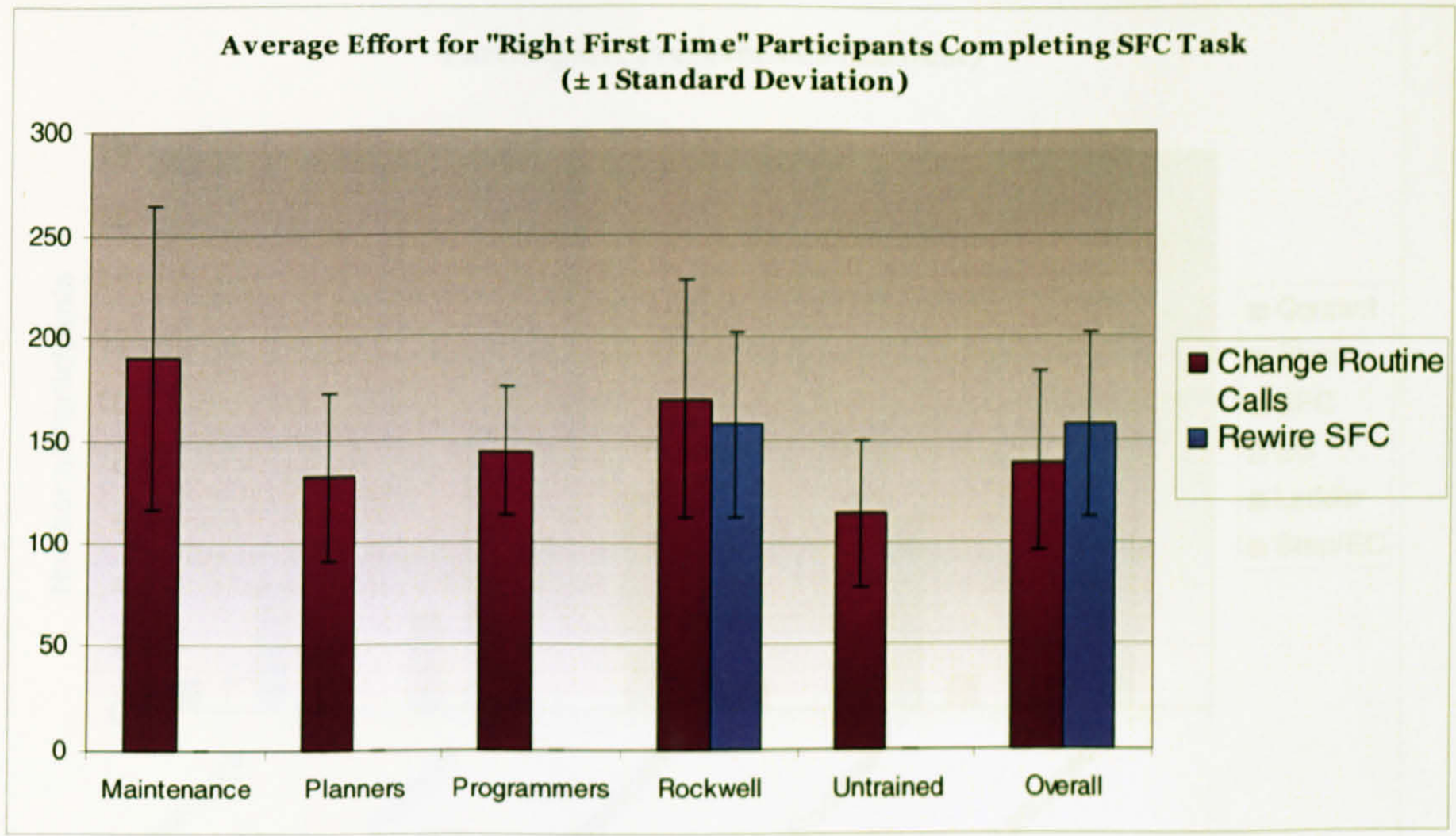

Figure 21 Average Effort for "Right First Time" Participants Completing SFC Task (± 1 Standard Deviation)

5.2.5 Impact of Prior Experience

The main value of taking both time and effort measurements is gained when these results are combined with participant choice for which tool was found easiest to use. This is based on the expectation that a participant will select as easiest the tool requiring the least amount of input from the participant, either in terms of time spent or physical effort required. Figure 22 shows the logic design tools found easiest to use by participants in each category, regardless of whether or not the task was completed correctly. This shows that there is a clear preference for EC in all categories except among programmers, with seven finding step ladder logic easiest to use, with one further programmer indicating a preference for ladder without differentiating between contact and step ladder structures.

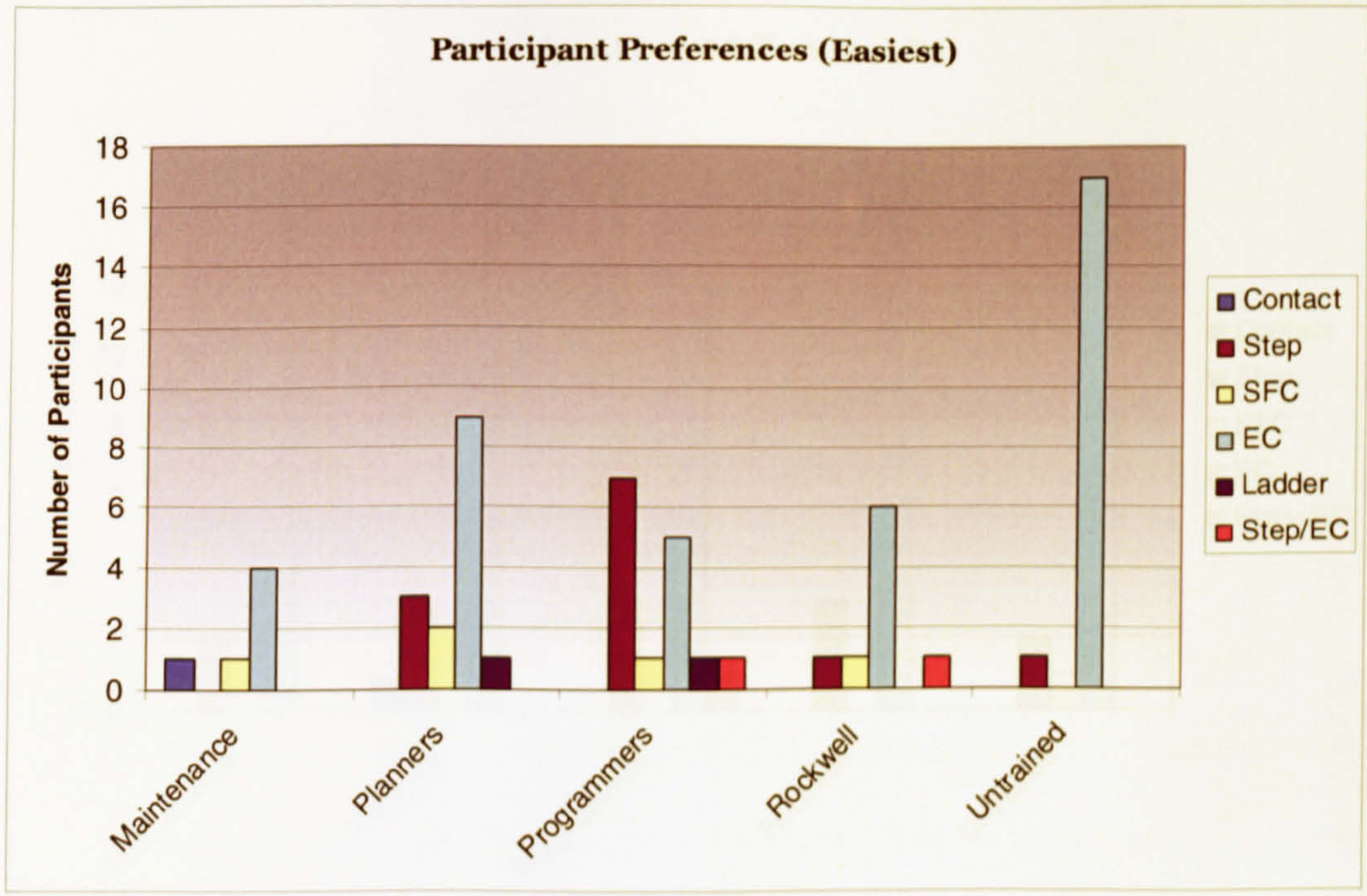

Figure 22 Participant Preferences (Easiest) by Category

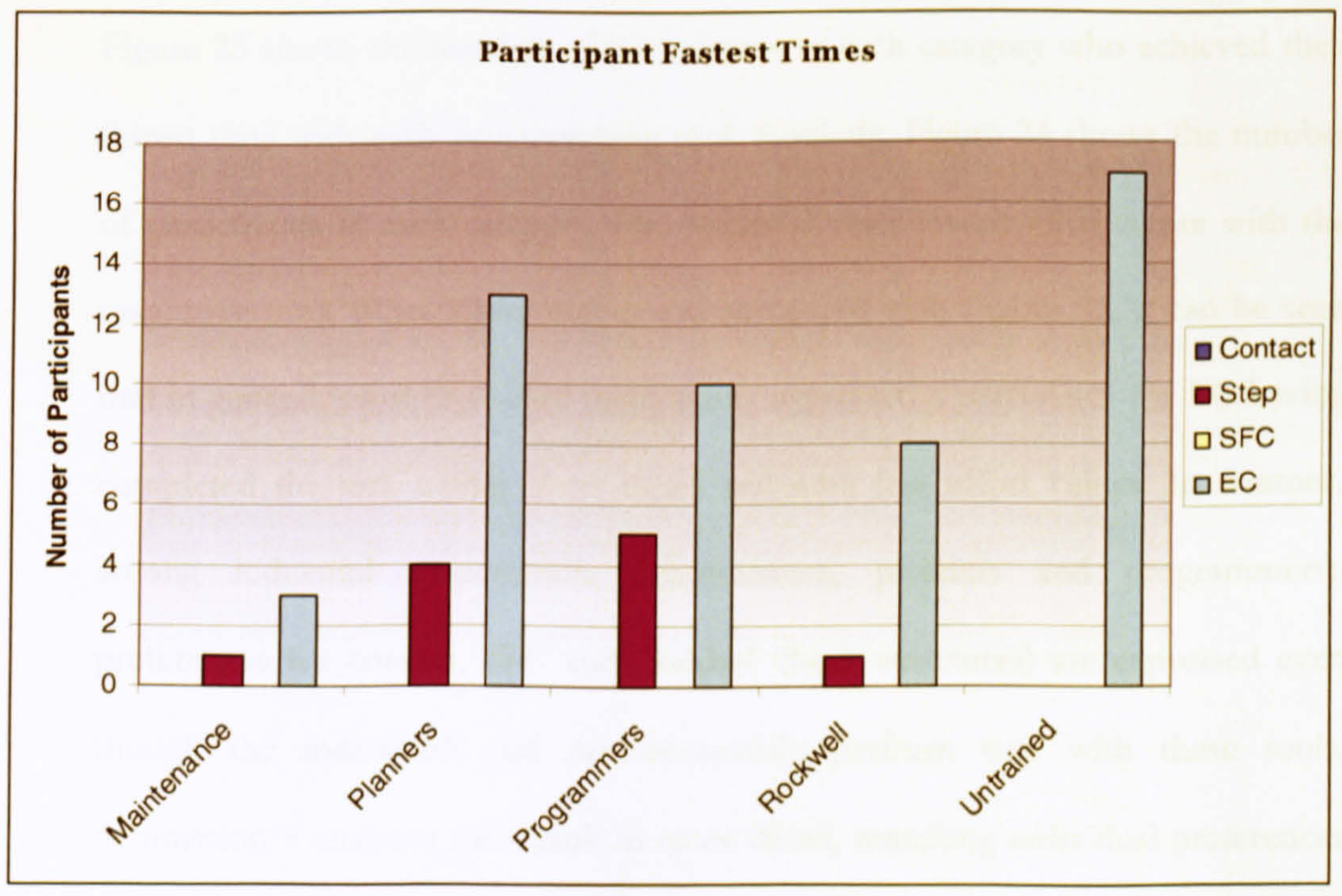

Figure 23 Participant Fastest (Correct) Times by Category

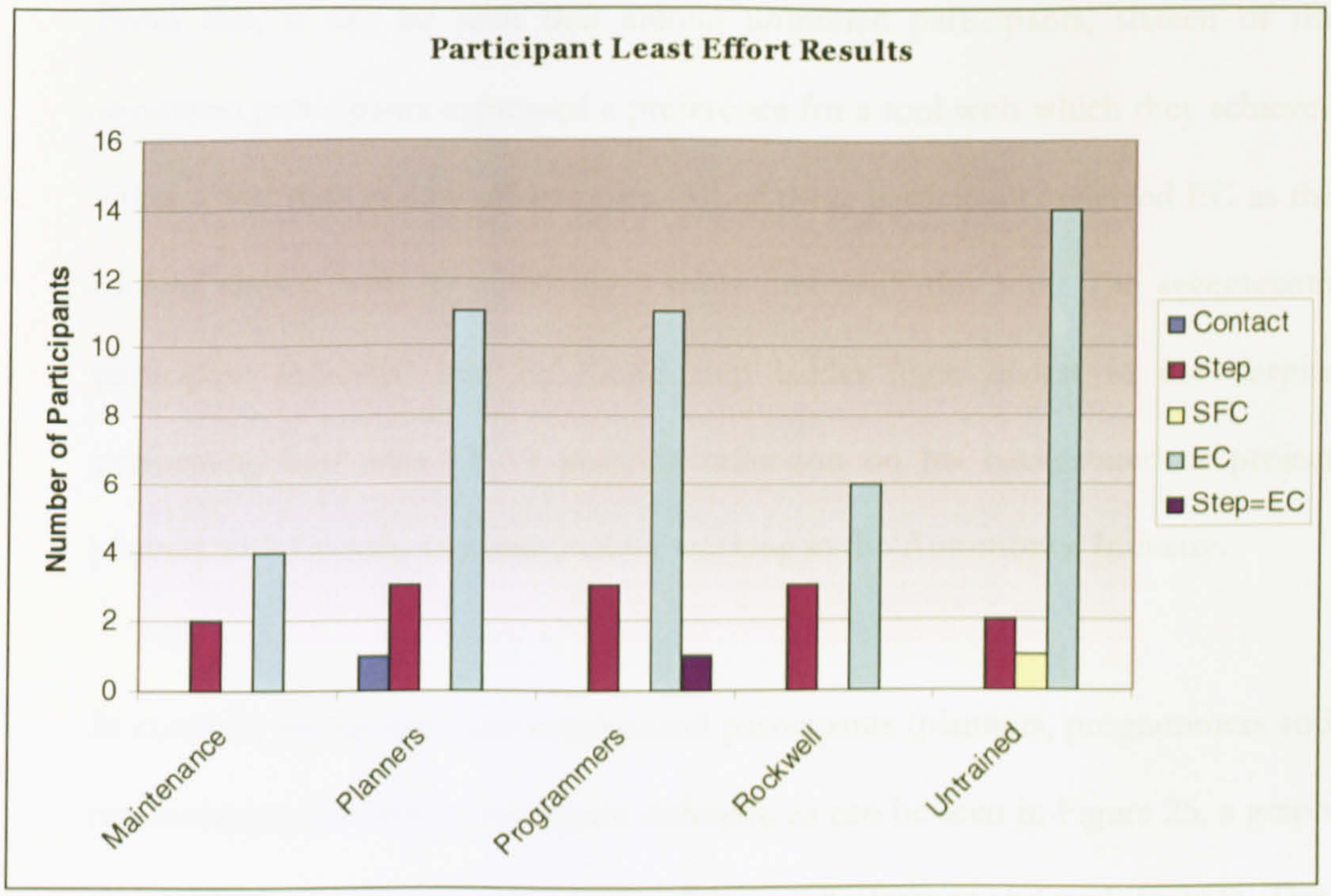

Figure 24 Participant Least (Correct) Effort Results by Participants

Figure 23 shows the number of participants in each category who achieved their fastest time with each programming tool. Similarly, Figure 24 shows the number of participants in each category who achieved their lowest effort score with the respective tool. When these graphs are compared with Figure 22, it can be seen that in general, most untrained participants expressed a preference for EC having completed the task within short times and with low effort values. In contrast, among industrial practitioners (maintenance, planners and programmers), preferences for contact, SFC and "ladder" (both structures) are expressed even though the individuals did not necessarily perform well with these tools. Submission 5 analyses this result in more detail, matching individual preferences to their performance.

From this, it can be seen that among untrained participants, sixteen of the seventeen participants expressed a preference for a tool with which they achieved either a fast time or low effort score. All of these participants selected EC as the tool of choice, with all achieving a short time with this tool. The seventeenth participant indicated that he found step ladder logic easiest to use despite performing best with EC, a possible reflection on his background as project planner with Comau, a system builder working in the Automotive Industry.

In contrast, among end-user experienced participants (planners, programmers and maintenance) the results were quite different, as can be seen in Figure 25, a graph derived from matches and mismatches between preference and performance. This shows that 25% of experienced participants selected a tool with which they had prior experience above those with which they achieved the lowest time and effort scores. In the case of 36% of participants it was not clear whether the choice was the result of performance or experience as their preference, prior experience and performance all matched. In a worst case scenario, it could be argued that 61% base their choice on prior experience rather than experimental results. 33% selected as easiest to use a tool with which they also scored well on both time and effort counts.

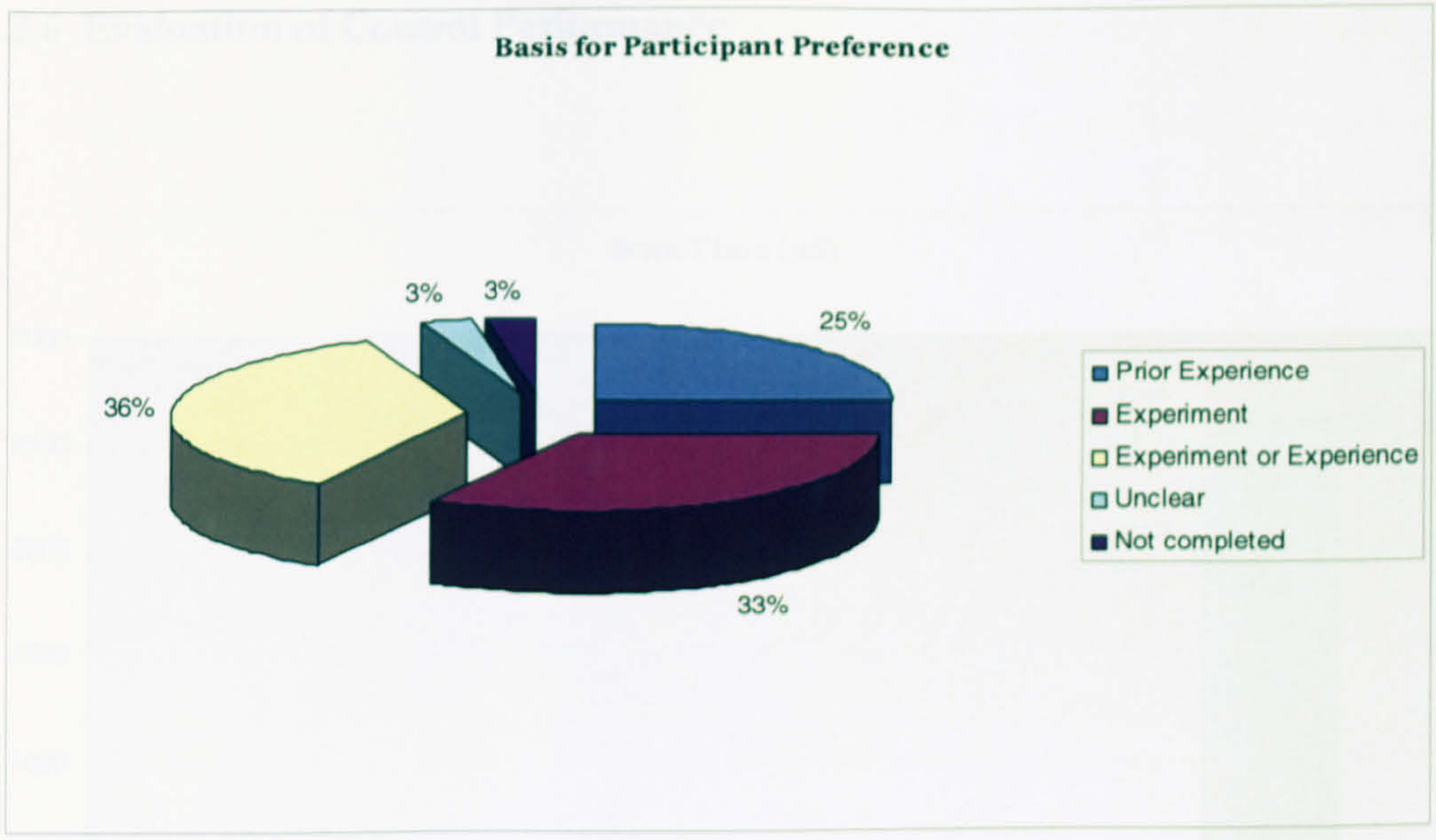

Figure 25 Basis for Participant Preference (Percentages)

5.2.6 Evaluation of Control Performance

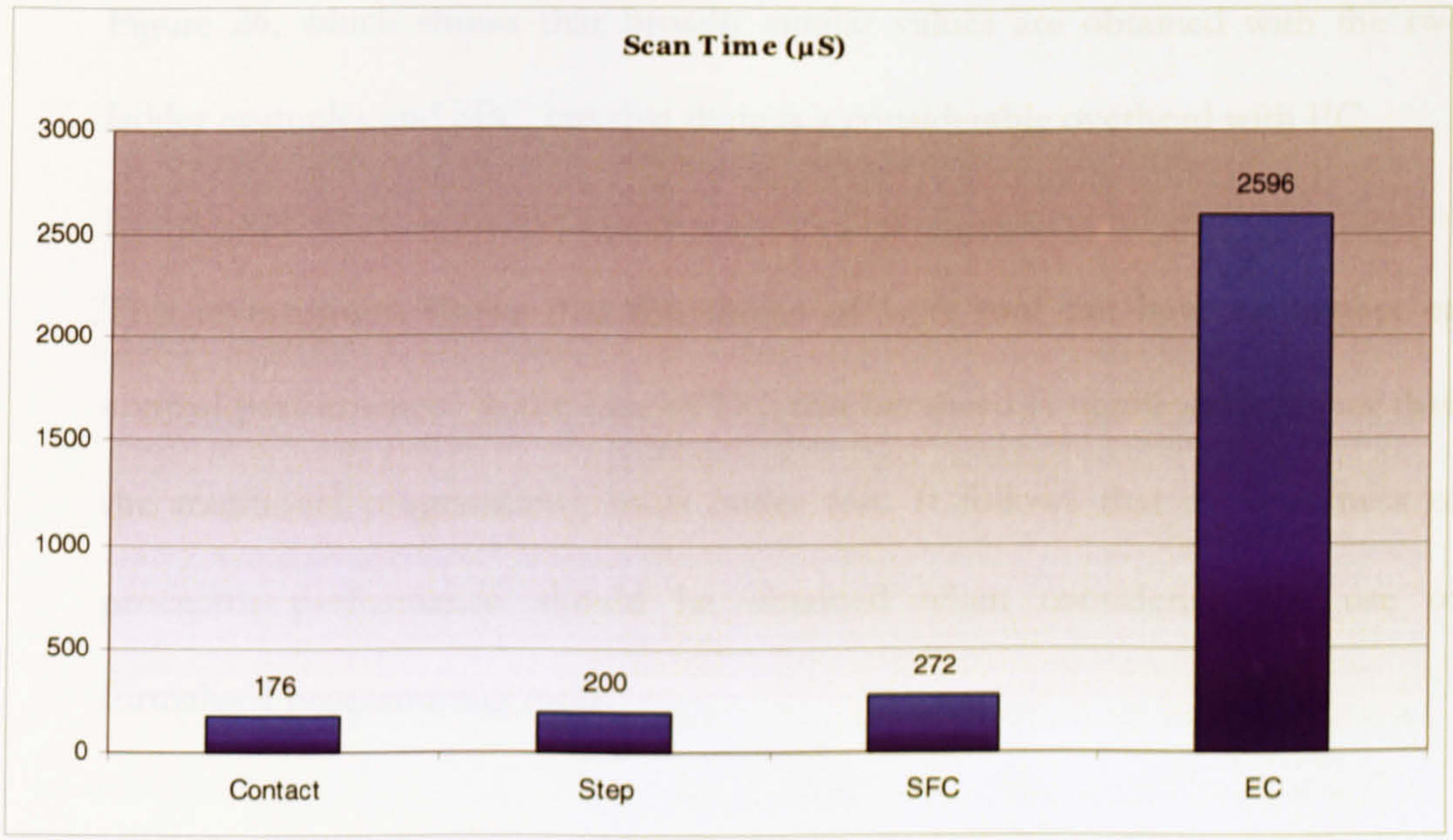

Figure 26 Scan Time Comparison

A popular mechanism for assessing the performance of a controller is the scan time. This is a measure of how long it takes for the processor to read data from inputs, analyse the algorithm and set appropriate outputs. The RS Logix 5000 programming tool allows PLC users to take readings of scan time. The value provided is rarely static owing to slight variations depending on the status of inputs, the parts of the code which are being analysed and the requirements for setting outputs. A short scan time is indicative of a processor delivering good control performance as this means that there is a very short delay between events occurring within the machinery under control and the required outcomes taking place.

In order to assess the differences between the four programming approaches under test here, the machine was set running with each of the programming tools

and left in this state for a period of five minutes. At the end of this spell, the maximum processor scan time was recorded. The values obtained are shown in Figure 26, which shows that broadly similar values are obtained with the two ladder examples and SFC, but that there is a considerable overhead with EC.

This investigation shows that the choice of logic tool can have an impact on control performance. In the case of EC, this overhead is significantly higher than the traditional programming tools under test. It follows that an awareness of processor performance should be obtained when considering the use of formalised programming tools.

5.2.7 Diagnostic Test

The aim of the diagnostic test was to identify the approach each participant used to detect the nature of generated faults on the system. The justification for this work is based on the difference in diagnostic approach between the traditional programming methods, in which programmers have the ability to access diagnostic data at the lowest level, and EC, in which this is partially obscured by the complexity of the underlying code. Thus the objective of this test served the following purpose: given a working HMI, what is the likelihood that a participant decides to obtain diagnostic information from code rather than through use of the HMI.

Various observations were made regarding detection of the fault. Many participants had a tendency to read the diagnostic message from the HMI screen and confirm its nature by looking at the code, as can be seen in Figure 27.

An interesting point to note is of all 63 tests, there were no instances where the diagnostic messaging was not updated with the SFC and EC examples. With the ladder programs however, there were a few instances (a total of 4 with contact ladder and 1 with step ladder) where the code was updated correctly giving a correct control function but diagnostic messaging code was not updated leading to the display of incorrect messages on the HMI screen. This provides further evidence to show that limitation of the tasks the programmer can do (EC) as well as hiding functionality in blocks (SFC) can help ensure that system changes are completed correctly together with appropriate diagnostic messaging.

Although not a full test of diagnostic capability, what this work does show is that diagnostic information can be obtained equally well from HMI systems as from code. Thus, the issue is not so much the ability to obtain the information, rather a question of ensuring that there is close collaboration between HMI and controller and building trust that the system provides sufficient information. The observation that many trained participants chose to check what was causing the message rather than relying solely on the text supports this view. This in turn suggests that as far as diagnostic function is concerned, the aim has to be to ensure that diagnostic messaging is correct. This will involve correct configuration at start-up and ensuring that changes do not lead to unforeseen results. EC has the advantage in the sense that programmers without a good understanding of the underlying operating code are not able to make ad-hoc modifications to code.

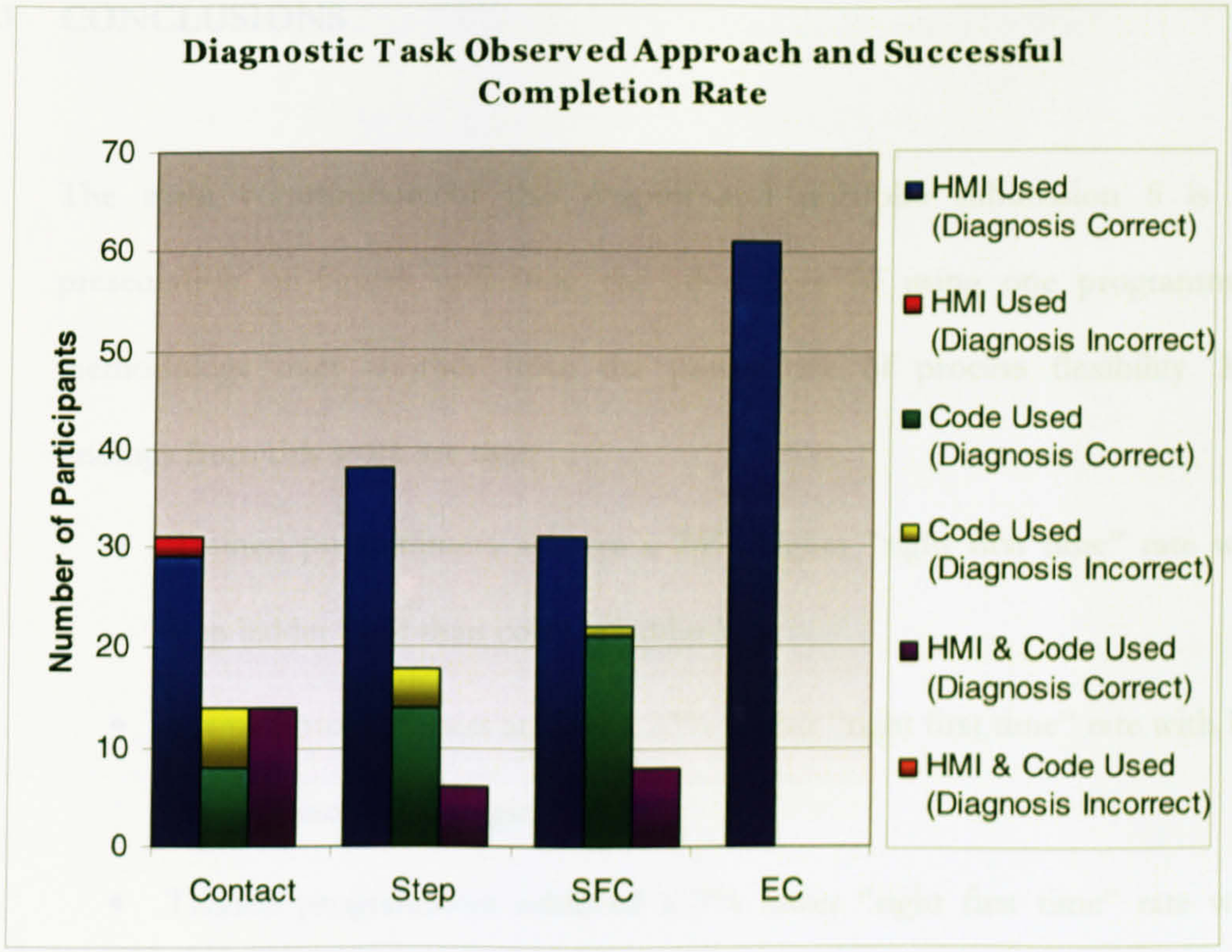

Figure 27 Observed Approaches to Diagnostic Task and Success Rate with Each Approach

Figure 27 shows how individual participants approached the diagnostic function for the task in which they were asked to deduce the reason why a particular machine was not working. For each programming methodology under test, it shows how many participants opted to achieve diagnosis of the fault through exclusive use of the HMI, those opting for exclusive use of code and those using a combination of the two, together with the number of participants who provided an incorrect response. In the case of EC, no code was visible so all participants were forced to rely exclusively on the HMI. It is clear from this graph that the use of an accurate HMI provides benefits in terms of ability to diagnose a fault effectively.

5.3 CONCLUSIONS

The main contribution of this chapter and portfolio submission 5 is the presentation of figures indicating the advantages of using one programming methodology over another from the perspective of process flexibility. Key findings from this work are that:

- Trained programmers achieve a 26% higher "right first time" rate with step ladder logic than contact ladder logic.
- Trained programmers achieve a 33% higher "right first time" rate with EC than contact ladder logic.
- Trained programmers achieved a 7% lower "right first time" rate with SFC than contact ladder logic.
- The structure of EC and step ladder allowed untrained participants to achieve "right first time" completion rates of 78% and 67% respectively, despite their limited prior knowledge of the tools.
- Differences in approach were seen with both step logic and SFC, with the chosen approach to the task having an impact on completion rate:
    - 30% improvement in "right first time" rate for step logic if step numbers are changed instead of code.
    - 56% improvement in "right first time" rate for SFC if routine calls are changed instead of rearranging SFC elements.
- Use of EC gives a 79% saving in process flexibility (time) over contact ladder, both overall and among trained programmers.
- Use of EC gives a 54% saving in process flexibility (time) over step ladder (37% among trained programmers).

- Use of step ladder gives a 54% saving in process flexibility (time) over contact ladder (66% among trained programmers).
- At least 25% of trained industrial practitioners select as their tool of choice the technique with which they had prior experience over that with which they performed best.
- In the application under test, contact ladder gives a 93% saving in processor scan time over EC.
- Approximately 50% of all participants chose to obtain (or confirm) diagnostic information from code rather than HMI, indicating the level of mistrust in the visualisation system.

None of the industrial participants mentioned that they had ever undertaken this type of evaluation in the past and therefore it is not thought that this type of experimental work is undertaken by industrial users of PLC's.

6.0 SUMMARY & FUTURE WORK

6.1 OVERVIEW

In response to a desire to encourage conservative users of automation products to adopt tools and techniques different from those used at present, an evaluation has been conducted which shows that there is a clear link between PLC logic design methodology and manufacturing flexibility. The principal achievement of this work is that it allows industrial practitioners to see that the choice of programming tool has an impact on cost in use, and therefore allows PLC users to understand that evaluation beyond the initial factors of purchase, training and support package costs is necessary when assessing a PLC or programming methodology for use in a project. It also highlights the need for the selection of objective measures when conducting this type of evaluation.

In order to achieve this result, work was conducted in a number of stages. Firstly, prior work in academia was reviewed but was found to use comparison methodologies which were not appropriate for delivering the goals of this project owing to the need for productivity to be inferred from measures of software complexity. Focus of this earlier work was primarily on program creation and the tools tested were found to lack industrial relevance. In order to overcome this, a set of semi-structured interviews were conducted, and the results of this investigation were used to formulate an experiment using a methodology which overcomes these limitations. Finally, experimental work was conducted to apply the comparison methodology which had been devised previously.

The interviews conducted among a number of car companies identified the existence of company standards with three unique structures. A significant contribution of this work is the identification of a link between chosen programming method and PLC vendor, following from the availability of better support for certain programming languages by some vendors than others. Country-specific preference of certain tools and techniques above others was found to be a natural consequence, also owing to the strength of automation vendor presence in a particular market, with German companies mostly making use of SFC on the Siemens platform and with British and US based operations opting for forms of ladder logic using Rockwell Automation products. Familiarity was identified as a factor influencing product choice, highlighting the need for expressing new product benefits in financial terms as the means for encouraging change. Although anecdotal evidence about these facts was seen, there was no prior published information of code structures used in industry. Therefore, a major achievement of this work is the capture and documentation of this information in a formalised manner.

6.2 INNOVATION

In order to overcome the limitations of previous work, the task-focussed evaluation of industrial programming techniques was proposed and conducted with a focus on the measurement of production flexibility. Unlike earlier work, this measured productivity and expressed it as a time saving, eliminating the need to infer business impact from technical measures. A further key differentiator was the direct measurement of user input to supplement time values allowing the measurement of two measures of performance unlike earlier work which focussed

exclusively on one parameter. The main characteristics of the methodology created for this comparison are summarised in Table 2.

	Industrial Approach	Academic Approach	Proposed Approach
Basis	Evaluation of package offered by vendors	Analysis of software	Evaluation of tool suitability from a process perspective
Measures	Purchase cost Training cost Familiarity Support Package Technical Features	Software Metrics (e.g. basic elements, logic expressions)	Direct measurement of manufacturing process specific measures: time and effort
Provides	Information about up-front costs and partial information about usage costs	Details about software complexity	Information about manufacturing process cost
Advantages	Straightforward application Easy to understand	Objectivity Rigour Vendor independence	Provides direct measure of productivity Allows continued use of existing measures Supports existing industrial approach Brings objectivity & rigour to industrial approach Wide applicability (e.g. program creation, program modification) Allows assessment of the impact of prior experience on tool selection
Disadvantages	Provides partial information about ownership cost Potential to be influenced by personal preferences and emotional factors	Limited in scope – can be difficult to obtain a fair comparison between disparate concepts Most suitable for evaluating program creation effort Programmer effort needs to be inferred Results difficult to understand	Need to design appropriate experiment Reliance on availability of skilled personnel Time consuming

Table 2 Existing and Proposed Mechanisms for Comparing PLC Software Structures

6.3 ACHIEVEMENTS

Conducting this experimental work has provided several significant results which can be seen as the consequences of the comparison method developed here. A result of particular interest is the finding that there are large differences in completion time and effort between the programming methods under test. Among skilled programmers, there is a 79% time saving through using EC over contact ladder logic, and a 37% time saving through using EC over step ladder logic. It can clearly be seen that the choice of an incorrect methodology can have a major impact on engineering cost. In parallel with this, the results show that there are significant differences in "right first time" completion rates between the tools. For example, among skilled programmers, there is a 67% "right first time" rate with contact ladder logic compared to a 100% "right first time rate" with EC. The work has also highlighted that some tools and techniques can be used very effectively by untrained programmers who are able to achieve fast times and high "right first time" rates despite having very limited prior knowledge of the tool, as can be seen with a 78% "right first time" rate among inexperienced participants. This also has a clear impact on the engineering cost of implementing a process change – both in terms of the length of time required to complete the task, and also in terms of the required skill level of the individual conducting the change.

The experimental work has also highlighted the impact of a poorly designed user interface on both performance and perception. The fact that the tool is difficult to use was clearly a major factor in the poor performance of participants with the SFC package. An appropriate method of verification as to whether usability is a function of the tool itself or simply the manufacturer's implementation would be

to conduct a cross-platform application of this evaluation. This could be achieved easily using the comparison methodology developed for this research. This result also highlights another advantage of the evaluation methodology: the use of a third-party to observe programmer performance yields additional information about how a particular tool can be used optimally.

A particularly interesting result obtained from this evaluation is the observation that participants perform far better with step ladder logic than contact ladder logic, showing that prior claims that ladder logic is difficult to use and inflexible should relate to the contact ladder structure and not the language as a whole.

Although the diagnostic evaluation conducted here was limited in scope it revealed that given that nearly 50% of participants opted to obtain diagnostic information from code rather than from the HMI (or opted to verify their answer in code) even though this information was readily available from the HMI. This indicates the level of mistrust in diagnostic systems. At the same time, it was found that participants were more prone to omit necessary changes to the diagnostic interface with the ladder-based examples than with the graphical programming tools, indicating that certain methods are more likely to facilitate the correct update of diagnostic information, and highlighting the need for the development of systems-based diagnostic systems in which control and visualisation functions are unified. The development of mechanisms for ensuring that diagnostic visualisation systems are updated accurately would make for an interesting area for further work.

A further advantage of the comparison methodology used here is its flexibility, allowing it to be applied to compare new programming tools and products against existing techniques, in turn allowing users to make a better informed decision of automation product (based on cost of use) and vendors to benchmark their new offerings against current products, and to develop a marketing strategy in which business benefit of investment in a product can be demonstrated to potential customers.

As well as the primary results outlining the differences between the programming tools, the capture of twin measures of performance together with participant preference allowed for the evaluation of whether there is a match between prior experience and performance, and in turn helped demonstrate that at least 25% of industrial participants (programmers and planners) selected their preferred tool as that with which they have prior experience, even though they performed better with another methodology on either time, effort or both measures. In contrast, virtually all untrained participants expressed a preference for a tool requiring a short time or little user input. No prior evidence, either published or anecdotal to show that this type of task-centred evaluation has been conducted by either end users or vendors was seen, nor is there any prior published material documenting the level of prejudice seen in the selection of automation software. The implication of this result is that automation end users should select objective measures which can be related to business when evaluating PLC logic design tools for use in their projects.

6.4 FUTURE WORK

Arguably, a limitation of this work is that it deals with only one part of the software management process: modification. Therefore, the next step from this work is to conduct similar comparison work to establish the strengths and weaknesses of logic design approaches for the creation of control programs. Combining these aspects together will provide a good overview of the desirable characteristics of industrial control software and can in turn stimulate the development of formalised programming models suitable for use in manufacturing automation. Aspects of particular interest and how the work conducted in this project helps to facilitate them are outlined here.

6.4.1 Diagnostic Mechanisms

It is clear from both the results of the survey and from the experimental work that the ability to obtain accurate diagnostic information from control systems is essential. The lack of trust in existing HMI systems is also evident. The code encapsulation provided by programming tools therefore is likely to be seen by end users as hindrance from a diagnostic point of view as it is harder for maintenance technicians to understand the operation of a system in full. This can be overcome through the development of mechanisms to demonstrate an effective and reliable link between control and diagnostic functions. There is therefore potential to create a mechanism for ensuring that the diagnostic function is always a correct reflection of the system control performance. The approach used in the experimental work is a valid starting point for work in this field.

6.4.2 Study of Logic Constructs

Commercial deployments of EC conducted to date have required significant amounts of engineering time at the outset of the project to create profiles to operate the machinery and mechanisms used in plant. This owes to the fact that the existing template supplied by the vendor is limited in scope, requiring a new library of devices to be created. A method for reducing this initial configuration cost would be through a study of the logic constructs used in automotive applications at present, and the development of a tool for building typical templates based on the findings. This is something which follows from the results of the investigative work conducted here, in which it was seen that the programming tools presenting a single point change provided the best results. Application of the same idea to the design of control functions will no doubt provide similar productivity improvements in the initial phase of a project.

6.4.3 Creation of Formalised Programming Tools

Preliminary indications of this work showed that although EC presented benefits in terms of ease of implementing process changes, there is an overhead in terms of processor and control performance. It follows that the challenge for automation vendors is the development of a formalised programming tool which has the usability and understandability of EC whilst matching the control performance of traditional programming tools. Possible mechanisms for achieving this are through a return to a compiled programming tool, or the creation of mechanisms to enable and disable steps in a SFC program, potentially building on S88 and phase manager tools used in batch programming packages.

Further work in the area of creating formalised programming tools (related to the points covered in 6.4.2) is the definition of vendor independent models for the standard control operations and functions used in industry. This is potentially a role best undertaken in academia in order to overcome commercial considerations.

6.4.4 Framework for Autonomous Intelligent Agents

In (Hajarnavis, 2005b), the requirements necessary in order to achieve autonomous agent-based control in a manufacturing environment are discussed. It was noted that one of the biggest challenges faced in realising this goal is effective object-orientation. Through addressing diagnostic mechanisms, the creation of a simple mechanism for creating logic constructions and addressing the limitations of existing programming tools, the definition of functional modules with a common interface can be achieved. These modules will in turn provide the starting point for control based on autonomous agents which are capable of providing the diagnostic information necessary in a manufacturing environment.

7.0 CONCLUSIONS

The main innovation provided from this work is the introduction of objective analysis into the field of industrial control systems in turn allowing the expression of software design methodology in terms which can be understood as business benefits. This is achieved through the creation of a mechanism for automation end users and suppliers to compare different and disparate programming tools and helps provide the following contributions to knowledge:

- Demonstration that the formalised tool under test required a lower skill level for successful completion than any of the other tools, with untrained participants able to achieve a 78% "right first time" rate compared to a 17% "right first time" rate with contact logic.
- Demonstration of the benefits of formalisation, in terms of a 33% improvement in "right first time" rate together with a 79% productivity saving over contact ladder logic (among programmers).
- Identification that the structure of contact ladder logic is the reason for the perceived inflexibility of the programming language.
- Demonstration of the impact of prior experience of a tool on product selection in measurable terms, with at least 25% of programmers and planners selecting a tool based on their prior experience rather than their performance.

Application of this approach in turn provides an understanding of the following areas which are of benefit to automation end users, none of which could be achieved prior to conducting this research:

- Demonstration of the impact of tool choice on manufacturing flexibility and consequent impact on the cost of implementing process changes.
- Demonstration of the impact of prior experience on tool choice, leading to reluctance to adopt and use newer tools.
- Identification of the need for conducting objective analysis when evaluating a tool.
- Presentation of a methodology which allows end users to select a tool most suited to flexible processes.
- Presentation of information which allows end users to reduce reliance on skilled programmers to implement process changes.
- Presentation of data which allows end users to see the impact of their choice of working practice on their productivity.

These points also provide the following additional benefits to automation vendors, specifically relating to the provision of a mechanism for automation vendors to test, benchmark and market their products against other techniques and to express the results in cost of ownership terms, providing in turn:

- An awareness of the needs, concerns and priorities of their customers in the automotive industry.
- An awareness of the need for full usability testing of programming tools prior to release, and impact of sub-optimal user interfaces.
- An awareness of the value of formalised programming tools.
- Ideas for how formalised programming tools could be developed further and used to support autonomous agent-based control.

Overall, the initial objective of identification of the logic design methodologies used in the car industry and evaluation of the impact of choosing a particular approach on manufacturing flexibility has been successfully achieved. The work is novel owing to the fact that no prior comparison had been conducted between industrial programming concepts. Innovation is demonstrated in the mechanism by which the results were achieved.

8.0 REFERENCES

BANI YOUNIS, M. AND FREY, G. (2004) Visualization of PLC Programs using XML, In *Proceedings of the 2004 American Control Conference*, Boston, Massachusetts, June 30- July 2 2004, pp3082-3087

CARD, S.K., MORAN, T.P. AND NEWELL, A. (1980) The Keystroke-level Model for User Performance Time with Interactive Systems, *Communications of the ACM*, Vol. 23 Iss. 7, pp396-410

CARPANZANO, E., CATALDO, A., AND TILBURY, D. (2004) Structured Design of Reconfigurable Logic Control Functions through Sequential Function Charts, In *Proceedings of the 2004 American Control Conference*, Boston, Massachusetts, June 30- July 2 2004, pp4467-4471

DAS, S.K. (1996) The Measurement of Flexibility in Manufacturing Systems, *International Journal of Flexible Manufacturing Systems*, Vol. 8, Iss. 1, pp67-93

DAVID, R. (1995) Grafcet: A powerful tool for specification of logic controllers, *IEEE Transactions on Control Systems Technology*, Vol. 3, No. 3, September 1995, pp253-268

EDAN, Y. AND PLISKIN, N. (2001) Transfer of Software engineering tools from information systems to production systems, *Computers & Industrial Engineering* Vol. 39 pp19-34

FREY, G. (2000) Automatic Implementation of Petri Net Based Control Algorithms on PLC, In *Proceedings of the 2000 American Control Conference*, Chicago, Illinois, June 28-30, pp2819-2823

HAJARNAVIS, V AND YOUNG, K.W. (2005a) Intelligent Agents in Manufacturing Automation, In *Proceedings of the IEE Seminar on Autonomous Agents in Control*, Bicester, UK, 10 May 2005, pp29-34

HAJARNAVIS, V AND YOUNG, K.W. (2005b) A Comparison of Sequential Function Chart and Object-Modelling PLC Programming, In *Proceedings of 2005 American Control Conference*, Portland, Oregon, June 8-10, 2005, pp 2034-2039

KIERAS, D.E. (1988) Towards a Practical GOMS Model Methodology for User Interface Design, In *Handbook of Human-Computer Interaction*, Elsevier Science Publishers, B.V., pp135-157

KIERAS, D.E. (1997) Task Analysis and the Design of Functionality. In *The Computer Science and Engineering Handbook*, CRC Press, 1997

LEE, J.-S., AND HSU, P.-L. (2004) An improved evaluation of ladder logic diagrams and Petri nets for the sequence controller design in manufacturing systems, *International Journal of Advanced Manufacturing Technology*, No. 24, pp279-287

LEWIS, R.W. (1998) *Programming industrial control systems using IEC 1131-3*, Revised edition, The Institution of Electrical Engineers