

http://researchcommons.waikato.ac.nz/

Research Commons at the University of Waikato

Copyright Statement:

The digital copy of this thesis is protected by the Copyright Act 1994 (New Zealand).

The thesis may be consulted by you, provided you comply with the provisions of the Act and the following conditions of use:

- Any use you make of these documents or images must be for research or private study purposes only, and you may not make them available to any other person.
- Authors control the copyright of their thesis. You will recognise the author's right to be identified as the author of the thesis, and due acknowledgement will be made to the author where appropriate.
- You will obtain the author's permission before publishing any material from the thesis.

A CASE Methodology Companion for Batch Manufacturing Control Software for Programmable Logic Controllers

A thesis

submitted in partial fulfilment

of the requirements for the Degree

of

Doctor of Philosophy in Management Systems

at the

University of Waikato

by

Douglas A. Troy

University of Waikato

1995

Abstract

An investigation of the effectiveness of a domain-specific computer aided software engineering (CASE) tool to support the software development life cycle of control software for programmable logic controllers (PLCs) for manufacturing is described. An approach to guide the development of domain-specific CASE tools is articulated. These ideas are combined with past research in automatic programming for PLC software development to produce a prototype CASE tool, called a methodology companion, that is used as the vehicle to assess the potential of the CASE technology to improve industrial practice and the effectiveness of the individual features incorporated into it. The assessment is carried out using a series of structured interviews with practitioners to obtain their informed judgment concerning the potential and limitations of the CASE technology.

The results indicate that appropriate CASE technology and standards offer significant potential to improve the production of software artefacts, to facilitate coordination and communication amongst individuals, and to provide an improved organisational infrastructure. Domain-specificity contributes strongly to the acceptance and ease-of-use of CASE by the users, enables organisational changes that improve the effectiveness of individuals, and offers the potential to improve the structure of an organisation. Model-based analysis offers the potential to significantly improve the quality of the specification and the software developed from it. For design maintenance and automatic code generation to be successful throughout the life cycle, the study revealed that additional CASE tool functionality is required that supports the testing and maintenance of the software in a domain-specific manner. In the domain of manufacturing control, support is required for incremental code generation and on-line modification of operational PLC software.

The investigation is guided by an approach called action research, wherein the researcher and a client organisation that exhibits characteristics of the problem of interest collaborate to set up an experiment to bring about improvements, carry it out,

and then assess its consequences. The action research process proved effective in facilitating the development of a CASE tool that is likely to be adopted in practice by permitting practitioners to influence the design of the CASE tool, as well as to participate in its assessment. This collaboration in the research process brought about changes in their perceptions of how CASE technology may be used to improve the process and products of the software development life cycle.

In conclusion, domain-specific CASE methodology companions have the potential to significantly improve quality in the specification and coding phases of the life cycle and can contribute to increased effectiveness of both individuals and the overall organisation. Design maintenance and automatic code generation are problematic and require additional automated aid to be accepted in practice. The action research approach is an effective method for investigating the potential and limitations of CASE technology in a field-based, industry setting.

Acknowledgments

I would like to thank my supervisor, Dr. Robert J. McQueen for his guidance in this research and his assistance in building the contacts with industry that proved so valuable in the project. I would also like to thank him, his wife Anita, and their children for housing and feeding me when I needed a place to stay.

I am very appreciative of the support of Anchor Products, Ltd., for providing a grant-in-aid to support this project, and for working in a partnership in this research. I particularly want to recognise the support if Mr. Ian Steele and Mr. David Southall and their staffs for their support of the research.

The University of Waikato and Miami University have provided the resources to pursue this project. The Computer Science Department at the University of Waikato provided the initial support through a visiting lectureship position. Miami University provided the necessary professional leave and additional travel support through the Philip and Elaina Hampton Fund for International Initiatives.

The assistance of Mr. Richard Littin, who provided programming support is also gratefully acknowledged.

I am appreciative of colleagues who have reviewed this work and provided valuable comments, including Les Foulds, Murray, Osama, Al, Ann, Yasser, Fazli, Don M., and Don B.

My wife Gail and my children, Matthew, Adam, and Andrew are to be recognised for their support and understanding while I was away from them.

Finally, the support of friends including Jack, Iris, Lloyd, Rosanne, Barbara, and Brian is gratefully acknowledged.

Table of Contents

LIST OF FIGURES viii

LIST OF TABLES x

CHAPTER 1 INTRODUCTION 1

1.1 RESEARCH PROPOSITION 1
1.2 CASE TOOLS 2
1.3 THE PROBLEM DOMAIN 3
1.3.1 Batch Manufacturing and Control Systems 3
1.3.2 Characteristics of Control Software in Manufacturing 5
1.3.3 Manufacturing Control Logic Example 5
1.3.4 Programmable Logic Controllers 7
1.3.5 Problems in PLC Programming 11
1.4 SUMMARY 14
1.5 OUTLINE OF THE DISSERTATION 15

CHAPTER 2 RESEARCH APPROACH 18

2.1 RESEARCH APPROACHES 18
2.2 ACTION RESEARCH 20
2.3 LOCUS OF INTERVENTION 23
2.4 SOFTWARE ENGINEERING PRINCIPLES 25
2.5 ASSESSMENT OF FINDINGS OF ACTION RESEARCH 26
2.6 APPLICATION OF THE RESEARCH APPROACH 27
2.7 RESEARCHER'S PRECONCEPTIONS AND INTERESTS 28
2.8 SUMMARY 29

CHAPTER 3 LITERATURE REVIEW 30

3.1 CASE ENVIRONMENTS AND TOOLS 30
3.2 DOMAIN SPECIFIC METHODS AND CASE 32
3.3 A GENERIC CASE ARCHITECTURE 37
3.4 GENERIC MODELS FOR REAL-TIME CONTROL 38
3.5 CASE TOOLS AND THEIR ABSTRACTIONS FOR REAL-TIME CONTROL 40
3.6 LANGUAGES AND CODE GENERATORS FOR PROGRAMMING PLCS 44
3.7 CASE TOOL ARCHITECTURE FOR PLC SOFTWARE DEVELOPMENT 49
3.8 AUTOMATIC PROGRAMMING 51
3.8.1 Reusable Software Components 51
3.8.2 Knowledge-Based Approaches 52
3.8.3 Fourth Generation Languages 53
3.8.4 Summary -- Automatic Programming 54
3.9 AUTOMATIC PROGRAMMING FOR PROGRAMMABLE LOGIC CONTROLLERS 54
3.10 MODEL-BASED ANALYSIS 55
3.11 DIMENSIONS OF CASE TECHNOLOGY 56
3.12 SUMMARY 59

CHAPTER 4 PROBLEM DOMAIN 61

4.1 INDUSTRY PARTNER 61
4.2 DEFINITION OF BATCH MANUFACTURING 62

4.3 A Representative Structural Model for Batch Processing Control.63
4.4 A Representative Language Specification of Batch Control Steps 66
4.5 Specification Language in Use at Anchor Products 69
4.6 Relay Ladder Logic Coding Conventions 77
4.7 Software Development Life Cycle 80
4.8 Analysis of Computer-Based Tool Support in Current Life Cycle 87
4.8.1 Domain Support 87
4.8.2 Analysis of Current Tool Functionality: Production Technology 89
4.8.3 Analysis of Current Tool Functionality: Coordination Technology 92
4.8.4 Analysis of Current Tool Functionality: Organisational Technology 94
4.9 Summary 95

CHAPTER 5 A METHOD TO GUIDE THE DESIGN OF DOMAIN-SPECIFIC CASE TOOLS 99

5.1 Point of Intervention and Definition of Roles 99
5.2 CASE Tool Development Methodology 101
5.2.1 Approach 101
5.2.2 Perform a Domain Analysis 102
5.2.3 Decide What Aspects of the Domain to Include in the CASE Tool 103
5.2.4 Identify the Abstraction Presented to the User 103
5.2.5 Decide What Aspects of the Abstraction that are Under the Control of the User 105
5.2.6 Define the Way the User Specifies the System Under Development 106
5.2.7 Define the Products of the CASE Tool 106
5.2.8 Implement the CASE Tool 107
5.2.9 Summary -- CASE Tool Development Methodology 107
5.3 Potential of User Collaboration in the CASE Tool Design 108
5.4 Research Proposition Elaborated 109
5.5 Summary 111

CHAPTER 6 A CASE METHODOLOGY COMPANION TO SUPPORT DEVELOPMENT OF BATCH CONTROL SOFTWARE 112

6.1 CASE Tool Architecture 112
6.2 CASE Tool Abstractions 113
6.2.1 Structural Model -- the Fixed Abstraction 114
6.2.2 Behavioural Model -- the Variable Abstraction 116
6.2.3 Abstraction Realisation 117
6.3 Specification and Manufacturing Rules 118
6.4 Functional Description Assistant 127
6.4.1 User Interface, Consistency Checking, and Help Functions 127
6.4.2 Additional Editing Functions 136
6.4.3 Report Generator: Design Analysis and Documentation 138
6.4.4 Model-Based Analysis Using Simulation 141
6.4.5 Automatic Code Generator 144
6.5 Summary 148

CHAPTER 7 ASSESSMENT 150

7.1 Approach 150
7.2 Expected Outcomes 150

7.3 USERS AND ORGANISATIONS SURVEYED 152
7.4 STRUCTURED INTERVIEW QUESTIONNAIRE 152
7.5 RESULTS OF THE INTERVIEWS 154
7.5.1 Production 155
7.5.2 Coordination 156
7.5.3 Organisational 158
7.5.4 People, Structure, Roles and Relationships 159
7.6 APPRAISAL OF THE THEORETICAL FRAMEWORK 161
7.6.1 Appraisal of Domain Specific Methods 162
7.6.2 Appraisal of Design Maintenance 163
7.6.3 Appraisal of Automatic Code Generation 165
7.6.4 Appraisal of Model-Based Analysis 165
7.6.5 Overall Acceptance 166
7.6.6 Appraisal of Life Cycle Changes 166
7.7 OTHER BENEFITS IDENTIFIED BY RESPONDENTS 167
7.7.1 Changes in Perceptions 167
7.7.2 Ownership of the Control System Becomes Organisation-Wide 168
7.7.3 CASE Tools Facilitate a Symbiotic Design Process 168
7.7.4 CASE Technology and Standards Support Outsourcing 169
7.8 SUMMARY 170

CHAPTER 8 CONCLUSION AND RECOMMENDATIONS 171

8.1 CONCLUSIONS 171
8.2 ADDITIONAL RESULTS OF THE INVESTIGATION 173
8.3 CHARACTERISTICS AND BENEFITS OF THE RESEARCH APPROACH 174
8.3.1 A Cyclic Process 174
8.3.2 A Learning Process 174
8.3.3 Incorporation of Management Decisions into CASE Tools 175
8.3.4 Collaboration with Practitioners Reduces Barriers to CASE Tool Acceptance 176
8.4 IMPLICATIONS OF FINDINGS FOR EFFECTIVE CASE TOOL USE IN THE DOMAIN 176
8.4.1 Effective Use of Model-Based Analysis may Require Changes in the System Life-Cycle 176
8.4.2 Design Maintenance Requires Additional Tool Support for Debugging and Maintenance 177
8.4.3 An Extended CASE Tool Architecture for Manufacturing Control 178
8.5 DIRECTIONS FOR FUTURE WORK 179

APPENDIX A SAMPLE FUNCTIONAL DESCRIPTION 182

APPENDIX B CONDENSED INTERVIEW TRANSCRIPTS 198

B.1 PRODUCTION 198
B.2 COORDINATION 219
B.3 ORGANISATIONAL 229
B.4 PEOPLE, STRUCTURE, TASKS 244

BIBLIOGRAPHY 251

List of Figures

Figure 1.1. Bottle-filling line state and control signals 6
Figure 1.2. Boolean Expression for Control of Bottle Filling Valve 7
Figure 1.3 Schematic Representation for Valve Control 7
Figure 1.4. PLC Architecture 8
Figure 1.5. Example of a Relay Ladder Logic code fragment 9
Figure 2.1. Cyclical Process of Research Approach. 23
Figure 2.2. Interacting Variables in an Organisation 24
Figure 3.1. Generic CASE System Architecture 37
Figure 3.2. Modelling Methods Suggested by Hatley and Pirbhai 39
Figure 3.3. Modelling Methods in STATEMATE 40
Figure 3.4. Examples of Sequential Function Charts; 46
Figure 3.5. Proposed PLC Software Development Environment. 50
Figure 3.6. Dimensions of CASE Technology Functionality 58
Figure 4.1. Model of Batch Manufacturing 62
Figure 4.2. Typical Process Step 63
Figure 4.3. Process Model for Batch Processing 64
Figure 4.4. Example of Operations, Phases, and Steps 65
Figure 4.5. Anchor's Control Specification Structure vs. Model Structure 70
Figure 4.6. Order of Step Initiation (from example in Appendix A) 72
Figure 4.7. Concurrent Execution of Steps 72
Figure 4.8. Structure of a Step 73
Figure 4.9. Simple Step and Control Commands 74
Figure 4.10. Flow of Control in Simple Step Example 74
Figure 4.11. Example of Conditional and Timer Commands from the Sample FD 76
Figure 4.12. Program Control Commands from the Sample FD 76
Figure 4.13. RLL Code Corresponding to Figure 4.11 77
Figure 4.14. RLL Code for Operates to Drive the Pump 79
Figure 4.15. Idealised PLC Software Development Life Cycle 81
Figure 4.16. Idealised Life Cycle Showing Participants 83
Figure 4.17. Maintenance and Troubleshooting Phase in Practice 85
Figure 6.1. Domain-Specific CASE tool architecture (FDA) 113
Figure 6.2. Standardised FD Structure Supported by the FDA 114
Figure 6.3. Structure of the Forms-Based Interface 115
Figure 6.4. Example of ASCII Import/Export Code for Allen-Bradley PLC 118
Figure 6.5. Specification and Manufacturing Dictionaries and Rules 119
Figure 6.6. Top Level Terminology Rule Base Dialogue Box 120
Figure 6.7. Terminology rule-base edit dialogue 120
Figure 6.8. Top Level Plant Item Type Dialogues 122
Figure 6.9. Top Level Plant I/O Edit Dialogues and Dialogue for Editing a Digital Item 123
Figure 6.10. Dialogue for Editing an Analogue Type 123
Figure 6.11. Generic RLL Template 125
Figure 6.12. Expanded RLL Code 126
Figure 6.13. RLL Templates for Item Types 126
Figure 6.14. User Interface Functions 128
Figure 6.15. Initial Screen After Opening an Existing FD 129
Figure 6.16. Header Dialogue Box 129

Figure 6.17. Description Dialogue Box 129
Figure 6.18. Real I/O Dialogue Box 130
Figure 6.19. Selection Dialogue Box 131
Figure 6.20. Step List Dialogue Box 131
Figure 6.21. Header Dialogue Box 132
Figure 6.22. Modification history dialogue box 132
Figure 6.23. Dialogue Showing Details of a Step 133
Figure 6.24. Precondition Dialogue Box 134
Figure 6.25. Operate Dialogue Box 134
Figure 6.26. Fault Check Dialogue Box 135
Figure 6.27. Editing using Copy, Paste, Find and Replace 136
Figure 6.28. Copy/Paste for Selections 137
Figure 6.29. Replace Dialogues 137
Figure 6.30. Find Dialogue 137
Figure 6.31. Design Analysis and Project Documentation Report Generator 138
Figure 6.32. Report Selection Menu 138
Figure 6.33. Messages Report 139
Figure 6.34. Selection Summary 140
Figure 6.35. Step Summary 140
Figure 6.36. Item/Step Cross Reference 141
Figure 6.37. Simulation Module 142
Figure 6.38. Selection States 142
Figure 6.39. Examining Step States 143
Figure 6.40. Forcing the State of Steps or Items 143
Figure 6.41. Updated State of Steps after Execution 144
Figure 6.42. RLL Code Generator Module 144
Figure 6.43. Configuration Menu for the Code Generator 145
Figure 6.44. Configuring Message Numbers 145
Figure 6.45. Configuring PLC Addresses for Software Tags 145
Figure 6.46. Message Numbers Automatically Assigned 146
Figure 6.47. Software Tag Addresses Automatically Assigned 146
Figure 6.48. Compile Menu 146
Figure 6.49. FD Fragment 147
Figure 6.50. RLL Code for Preconditions and Checks 148
Figure 6.51. Operates code for VP devices 148
Figure 7.1. Expected Software Life Cycle 151
Figure 7.2. Enclosed Areas Indicate Problem Areas in the Anticipated Life Cycle 167
Figure 8.1. Extended CASE Architecture to Support Maintenance and Debugging 179

List of Tables

Table 2.1. Basili's Improvement Oriented Paradigm 26
Table 4.1. Examples of Device Driving Commands 67
Table 4.2. Examples of Continuous Control Commands 68
Table 4.3. Effort In Life-Cycle Phases for Development 86
Table 4.4. Software Tools Utilised in the PLC Software Life-Cycle 87
Table 4.5. Problems in the Current Environment and Concepts to Address Them 97
Table 6.1. Terminology Rule Base Entry 119
Table 6.2. Plant Item Type Rules 121
Table 6.3. Example of Plant I/O Rules 122
Table 6.4. Examples of RLL Template Expansion Tokens 125
Table 7.1. Interview Results for CASE Tool Effectiveness 155
Table 7.2. Interview Results Concerning the Framework of Ideas 162

Chapter 1 Introduction

This chapter introduces the research topic, motivates the investigation, and defines terminology and concepts that are central to the work. The chapter concludes with an outline of the remainder of the dissertation.

1.1 Research Proposition

The thesis of this investigation is that computer aided software engineering (CASE) technology that is domain specific, incorporates model-based analysis, automatic code generation, and an appropriate methodology and standards, will be effective in supporting the software development life cycle for batch manufacturing control software for programmable logic controllers (PLCs).

The motivation for the investigation is as follows. Although CASE tools show promise for improving software development practice, past research reveals surprisingly limited use of CASE for the specification phase and that the overall impact of CASE tools on productivity and quality is unclear. Also, it will be shown in Section 1.3 that software development for PLCs poses particular problems, both due to its real-time nature and because of current industrial practice. This research aims to determine whether CASE tools would provide effective support for the design, coding, testing, and maintenance of PLC manufacturing control software. Thus, it aims to further research in the development of manufacturing control software as well as to investigate the potential effectiveness of CASE tools on industrial practice.

Specific recommendations from the literature concerning appropriate CASE tool features (domain-specific methods, automatic code generation, model-based analysis, methodology and standards enforcement) are identified in Chapter 3 as candidates for producing tools that will be effective and adopted in practice. An approach to guide the development of domain-specific CASE tools is proposed. These ideas are combined with past research in automatic programming for PLC software development to produce a prototype CASE tool system which is used as the vehicle to assess the

potential influence of the CASE tools on industrial practice and also to assess the effectiveness of the individual features incorporated into the CASE tools.

To obtain a realistic assessment, the action research approach is used as the research methodology. This methodology provides an industry-as-laboratory environment that exhibits the problems of interest, provides a mechanism to obtain knowledge through the assessment of the work, and an appreciation of its significance. To determine how and why the CASE tools are (or are not) effective, expected changes in the way the organisation develops control software are postulated, based on the literature, and these are compared with actual findings. The results are used to support or revise the underlying theory.

This research thus extends prior work in software development environments for PLCs, proposes a methodology to guide the development of domain specific CASE tools, identifies potential organisational outcomes resulting from the application of CASE tools in industry, and assesses principles that have been put forth in the literature as candidates for improving the effectiveness of CASE tools. Moreover, a unique prototype CASE system is developed to support batch process manufacturing.

The remaining sections of this chapter introduce CASE tools, software development environments, and the domain of control software for programmable logic controllers.

1.2 CASE Tools

Computer Aided Software Engineering (CASE) is the use of computer-based support in the software development process and is targeted toward those who perceive automation to affect the timing, cost and quality of the software development process and product (Brown, 1994; Henderson, 1990). Individual CASE tools are designed to aid or automate a specific task or activity in the software life cycle and include programs such as editors, programming tools, and many others that are further discussed in Chapter 3 (McClure, 1989; Fuggetta, 1993).

Despite the promise and variety of CASE tools, research shows surprisingly limited use in the area of requirements specification (Potts, 1993). Also, the impact on the

productivity of software developers and, ultimately, on the time-to-market of software is unclear. Some studies have shown improvements in productivity, while others have failed to identify a significant effect on productivity and only weak improvements on quality (Brown, 1994; McLendon, 1992; Henderson, 1990; Norman, 1989; Lempp, 1988; Card, 1987). Moreover, there is little agreement to be found concerning the best CASE tools, methodologies, or development paradigms (Kumar, 1992; Wood, 1989). Although it is unclear whether any available tool or environment technology will improve an organisation's ability to build and maintain software, application of software tools provides a mechanism to investigate technology to help to gather a baseline of productivity and quality data from which to judge future efforts, as well as to help to identify suitable metrics for the field (Brown, 1994; Kemerer, 1989).

An aim of this research is to investigate aspects of a CASE-based development environment that will address the problems of producing manufacturing control software and to maximise the chances that the work will transfer to industrial practice, in contrast to the current lack of CASE tool adoption. The next section introduces the domain of manufacturing control software and its characteristics.

1.3 The Problem Domain

This section introduces concepts and terminology concerning the domain of automated control of manufacturing processes and describes the growing role that computers and software play in manufacturing systems. In particular, the role of programmable logic controllers and their associated software is described.

1.3.1 Batch Manufacturing and Control Systems

Industrial manufacturing processes can be generally classified as continuous, discrete, or batch depending on whether the output product from the process appears in a continuous flow or in discrete batches or quantities (Fisher, 1990). The output from a continuous process appears in a continuous flow; in discrete manufacturing products are traditionally manufactured in discrete lots; and batch processes are discontinuous having characteristics of both discrete and continuous processes. A process is

considered to be batch in nature if, due to the physical structuring of the process equipment or other factors, the process consists of a sequence of one or more steps that must be performed in a defined order. For example, in a typical batch process, manufacturing ingredients are sequentially prepared (mixed, cooked, reacted, finished) and then packaged (Fisher, 1990; Shaw, 1982).

Control systems are either regulatory, discrete, or sequential (Fisher, 1990). Regulatory controls are used to maintain the outputs of a process as close as possible to some set point value despite influences of changes and disturbances. Examples are control of temperatures or flow rates. Discrete controls are used to maintain the outputs of a process to a target value chosen from a set of known (discrete) states. Sequential controls are used to sequence a process through a series of distinct states. Although control systems for batch manufacturing combine these three types of controls, most batch control systems require significantly more sequential and discrete control than regulatory control (Fisher, 1990).

Computer-based control technology is used as the means of automating the necessary schemes for real-time control of manufacturing devices (such as pumps, valves, and motors and to monitor sensors such as temperature, pressure, volume, and flow rate) that are used in batch manufacturing factories. Although advanced technology and improved production techniques have resulted in a reduction of the price of computer hardware for automation, the same is not true for software. The cost of application software has become a major consideration in automating batch processes (Fisher, 1990). In addition to the cost of control software, the major and growing importance of software in modern manufacturing has been recognised by researchers in academia, industry, and government agencies (Chaar, 1990). Roberts notes that the National Institute of Standards and Technology (NIST), which is under the auspices of the US Department of Commerce, places integrated control of production operations as one of the highest impact technologies in the manufacturing industry (Roberts, 1990). The characteristics of control software are described in the next section.

1.3.2 Characteristics of Control Software in Manufacturing

Some of the characteristics of device-control software are: 1) It is value and behaviour oriented: It must record data values as well as control the device's behaviour. 2) It must satisfy real-time constraints: It must record data when they are available and ensure that the device performs the right activities at the right times. 3) It involves both concurrency and distributed computation: Often, multiple devices must be controlled at the same time, and the devices are often physically distant from the computers. 4) It is of moderate size: It is better characterised as programming in the small than as programming in the large (Barstow, 1991). Harel includes device control software in a larger category called reactive systems which are systems that continually interact with their environment (Harel, 1992; Kurki-Suonio, 1994). Reactive systems include many kinds of embedded, concurrent, and real-time systems, but excludes data-intensive ones. The development of reactive systems has been found to be particularly problematic, posing some of the greatest challenges in system development (Harel, 1992; Abbott, 1993).

Thus, the investigation of improved tools and methods to support the development of control software for manufacturing is not only a theoretical concern, but also a practical one as well. The next section illustrates an example of a simple manufacturing situation and the its control requirements.

1.3.3 Manufacturing Control Logic Example

The following example of control logic for factory automation is adapted from (Halang, 1992). Consider the bottling line in Figure 1.1 which has the following state signals (outputs from the bottling line) and control signals (inputs to the line):

- p, a state signal that indicates that a bottle is in position for filling; it is closed when the sensor is depressed.
- s, a state signal from a switch that is set to indicate the bottle size; it is open when the line is filling bottle size 1 and closed when the line is bottling size 2 bottles.

- w1, a state signal produced by the scale when the bottle weight is less than weight 1, otherwise it is off.
- w2, a second state signal produced by the scale when the bottle weight is less than weight 2, otherwise it is off.
- r, a control signal to the bottling line that releases the next bottle from the supply.
- v, a control signal to the line that drives the bottle filling valve.

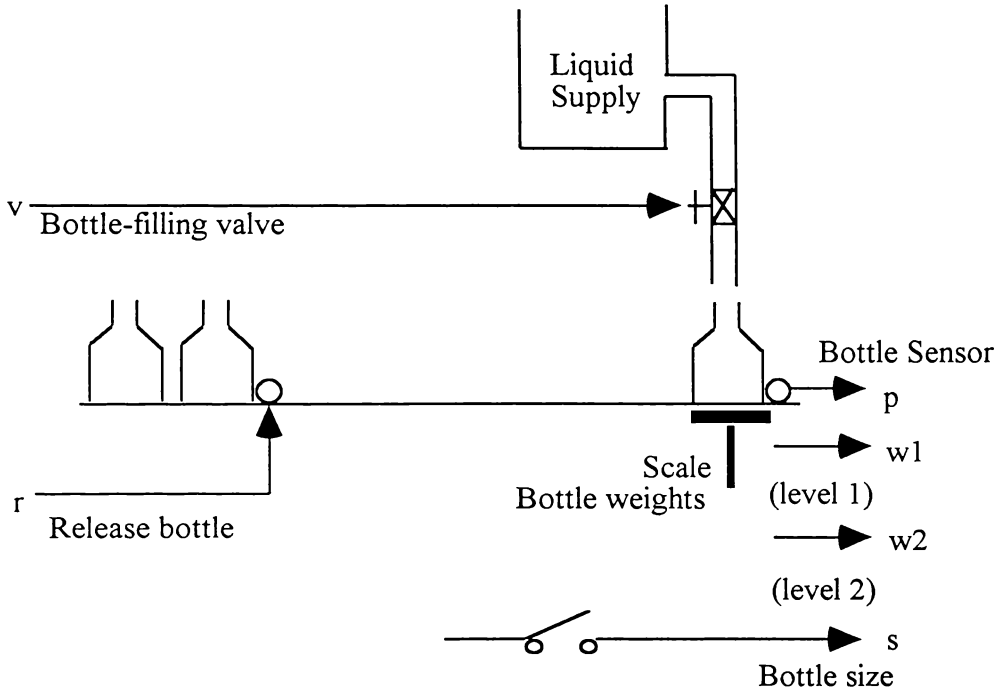


Figure 1.1. Bottle-Filling Line State and Control Signals (Halang, 1992).

The signal r to release a new bottle should be triggered by the release of a full bottle (signal p), ie., when switch p opens. The control of signal v (open valve) should be enabled when:

(1) the switch p is closed, **and**

(2) w1 is enabled by the scale (bottle contains less then weight 1), **or**

(3) switch s is closed and w2 is enabled by the scale (bottle size 2 is being filled, **and** contains less than weight 2)

The following boolean expression represents this logic:

v = p AND (w1 OR (s AND w2))

Figure 1.2. Boolean Expression for Control of Bottle Filling Valve.

An equivalent schematic representation is shown in Figure 1.3.

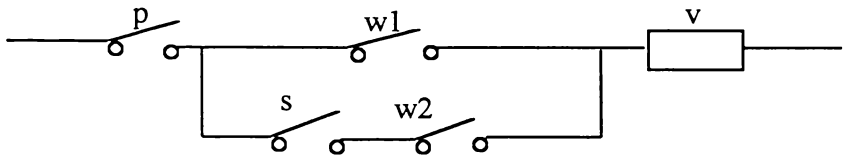


Figure 1.3. Schematic Representation for Valve Control (Halang, 1992).

A hardware circuit could be constructed using electromechanical relays or solid state logic gates to implement the logic shown above. The advantage to a control circuit implemented in hardware is the speed of signal processing: all inputs and outputs are processed simultaneously. The disadvantage is that the hardware circuit is inflexible; any modification to the control scheme requires a modification to the control circuit (Halang, 1992).

Another approach to implementation of control logic is to use *programmed logic*. In place of the hard-wired control circuit, a programmable processor is used which accepts the binary state of the state signals, evaluates the required boolean logic, and outputs the evaluated binary control signals to the process under control. The boolean logic is represented by a program in software and the processor is called a *programmable logic controller* (PLC).

1.3.4 Programmable Logic Controllers

The predominate control device in automated factories is the programmable logic controller (PLC) (Neidert, 1991). A PLC is a computer that simulates the behaviour of an electromechanical relay-based control system (Cassell, 1983). Figure 1.4 illustrates the architecture of a PLC. In place of hard-wired relay circuits, the PLC uses programmed logic in software. Inputs to the PLC consist of sensor inputs, such as limit switches and temperature sensors, both digital and analogue. Output signals are sent from the PLC to devices such as motor starters and valve actuators. During execution,

the PLC is dedicated to the continuous, repetitive task of examining system inputs, solving the current control logic, and updating system outputs (Halang, 1992; Johnson, 1987).

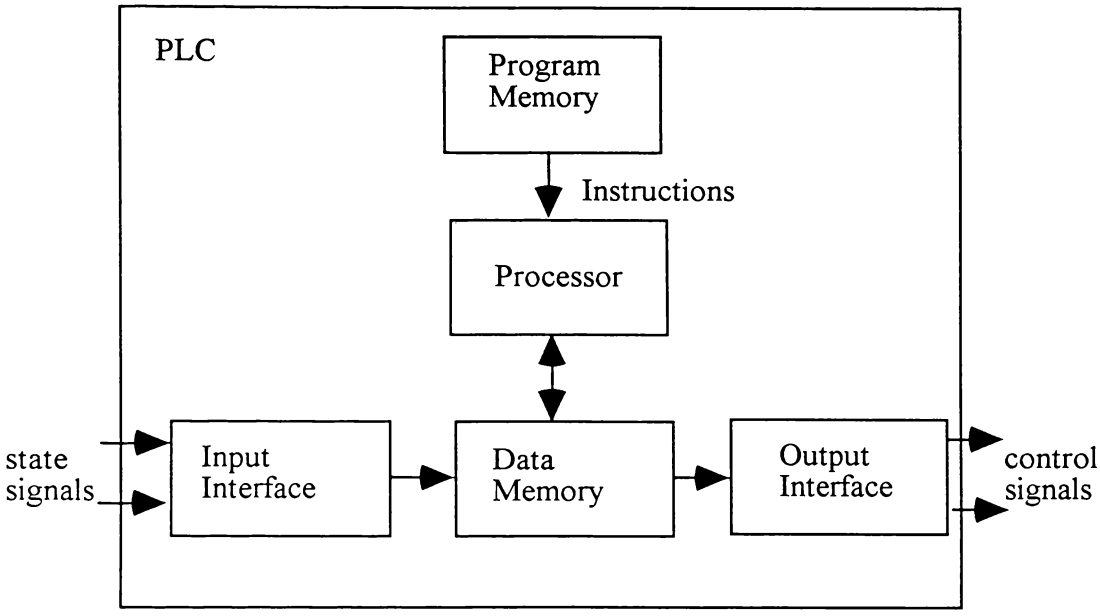


Figure 1.4. PLC Architecture (Halang, 1992).

Relay ladder logic (RLL), developed in 1969, is the predominate language for programming PLCs (Chaar, 1990; Johnson, 1987; Gayman, 1988; Roberts, 1990). RLL is graphic and resembles electrical logic diagrams used by electricians for wiring electromechanical relay circuits. The logic for the bottle-filling line discussed above is shown in Figure 1.5, expressed in typical ladder logic notation.

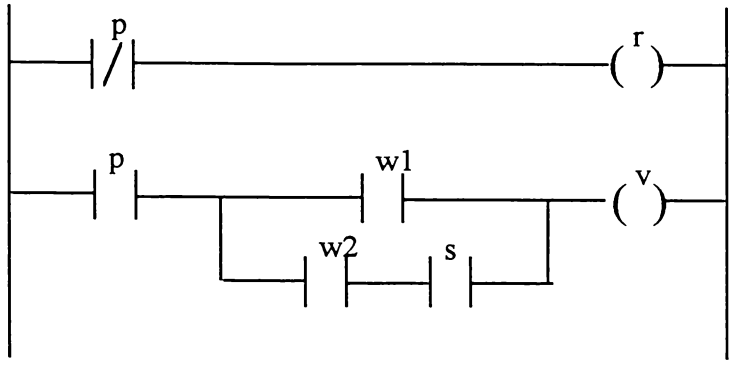


Figure 1.5. Example of a Relay Ladder Logic Code Fragment.

The vertical bars are called rails and the horizontal bars are called rungs. Power is conceptualised as flowing down the left rail and across the rungs toward the right rail. Inputs, or contacts, are pictured on the left side of the rungs and outputs, or coils, are shown on the right side of the rungs. In Figure 1.5, the top rung shows a normally closed contact for the input signal p, that is, the coil, r, is not energised when p is closed. When p is "open" the coil r is enabled. The second rung represents the boolean expression shown in Figure 1.2. The first contact is a normally open contact for input p. Power does not flow through this contact if p open, but will flow through when p closes. The branch in the second rung illustrates a boolean OR condition; the power is conceptualised as flowing through both branches. Thus, when contact p closes AND signal w1 is enabled then the valve signal v will be enabled, OR when contact p closes AND signal w2 is enabled AND switch s is closed then the valve signal v will be enabled.

PLCs are typically programmed using the PLC manufacturer's proprietary programming environment that operates on a personal computer (PC). Such programming environments include a graphical editor that allows a software engineer to create RLL diagrams which appear on the PC screen in a fashion similar to Figure 1.5. Completed RLL programs must then be up-loaded (again using the manufacturer's programming environment) into the PLC for execution. Once the RLL code is up-loaded, the PLC can be started and then the PC can be used to monitor the execution of the RLL.

Industrial PLCs have much more power than that illustrated in this example. Although PLCs were originally designed to deal with two-state binary and pulse signals, they now offer the possibility of implementing not only boolean functions but may process analogue signals, and may include timers, counters, numeric and string processing, and other features found typically in general purpose computers. Depending on the type of PLC, the number of inputs and outputs can range from 16 on small controllers up to 16,000 on large systems.

Due to the potentially harsh conditions in which they may operate, PLCs typically contain neither an operator interface nor mass storage devices such as disk drives. The configuration information and software are up-loaded into the PLC from a programming environment (on a personal computer) and then the software resides in the PLC's main memory and operates autonomously without human intervention (Halang, 1992). Operator interfaces, when employed, are implemented in separate systems that communicate with the PLC, periodically polling it concerning its state, and may have the capacity to send commands to the PLC.

RLL was an ideal PLC programming language for use by electricians because it resembled closely their traditional circuit diagrams, and for small control applications RLL programs were manageable. It provided a natural migration path for electricians from electromechanical circuit design to programming control logic with PLCs. Also, electricians could easily utilise the RLL diagrams for troubleshooting and diagnosing failures. PLCs have also been programmed with, but less commonly than RLL, boolean logic, sequential function charts, structured text, and instruction list languages.

The international community has recognised the need to standardise the programming languages used for PLCs. The International Electrotechnical Commission (IEC), Subcomittee 65B (formerly SC65A/WG6), has proposed a standard, known as IEC Standard 1131 for programmable controllers (IEC, 1991). Part 3, programming languages, defines four languages. Two are textual languages, Instruction List and Structured Text and two are graphic languages, Ladder Diagram and Function Block

Diagram. IEC 1131 also defines a graphical language for structuring the internal organisation of a PLC program called Sequential Function Charts (SFCs), also known in the industry as Grafcet. Grafcet, inspired by Petri nets with a few differences (David, 1992), was developed first as a French standard, then as an international one in 1987 (IEC, 1988). These languages are described in more detail in Chapter 3.

1.3.5 Problems in PLC Programming

The current state of control programming, and programming with PLCs, has numerous problems which are summarised as follows.

- Increasing Complexity of Software. PLCs are capable of handling considerably more I/O, have more computing power, and are capable of more higher level functions than when they were introduced two decades ago. An increasing number of projects are failing because of an inability to properly manage this complexity using current design and development practices (Neidert, 1991). This problem is compounded by the fact that RLL code is low level (Roberts, 1990). The structure of RLL programs can become hard to follow, and it becomes difficult to determine operations that occur in serial and those that execute in parallel. This problem becomes more pronounced as PLCs are applied to larger and more complicated tasks, resulting in larger and more complex RLL programs.

- Difficulty of Design Maintenance. RLL is poorly coupled to the design phase of the software life cycle (Roberts, 1990). Planning the sequence of operations of manufacturing devices and scheduling the execution of this sequence have traditionally been carried out independently, prior to developing the control software of manufacturing (Chaar, 1991). The planning activity is the domain of a process or manufacturing engineer, while the development of the control software is the domain of a software engineer or electrician. Process or manufacturing engineers, who are responsible for planning and specifying the manufacturing processes, do not write in RLL. In many organisations the output of the planning activity is a specification consisting of a combination of graphical plant design,

natural language descriptions, and/or logic or flow diagrams, typically produced using generic drawing and office automation tools. The output of the software development and implementation is a set of RLL programs produced using the PLC vendor's proprietary programming software. This necessitates a transformation from the process engineer's specification into RLL software by a software engineer. This transformation introduces opportunities for errors and, when the software has to be altered during testing or factory maintenance, the original design specifications are rarely updated (Higginson, 1990). Finally, if updates to design specification are performed, they are often weeks behind the RLL software changes (Larson, 1985). Thus original design documents are often inconsistent with the actual operation of the manufacturing system. Updating the design information is as important as revising the software itself since the lack of design maintenance contributes to the difficulty of software maintenance (Baxter, 1992). It would be more satisfactory to avoid, or reduce, manual translation steps (Busby, 1992).

- Lack of Methodology and Standard Support. As will be shown later in this investigation, the analysis and testing of the logic of the process engineer's specification is deferred until after it has been translated into RLL and often the specifications themselves are not standardised. This is due in part to the lack of automated tools that help to control the production of the specification to a standard and an absense of tools that support analysis and inspection of the design. Generic software tools (often generic office automation tools) do not facilitate developing control specifications, nor can they control the production of the specification to a standard. According to McClure, for CASE to improve software development productivity, the tools must facilitate following the steps and rules of a methodology and automate control of what the developer can do (McClure, 1988). This lack of support causes two problems. First, control specifications may not be standardised between process engineers placing burdens on users of the specifications. Second, problems in the specification often go undetected until late in the life cycle, where they are costly to fix.

- Lack of Software Portability. In spite of the recently introduced standards for PLC programming languages, there is no accepted and implemented standard for RLL in industry, which has resulted in each PLC vendor providing similar, but incompatible versions of RLL. Thus, RLL software is not portable between various makes and models of PLCs. In many installations, several different variations of PLC hardware and associated RLL languages are in use. This demands that programmers must be familiar with many types of hardware and their software variations (Jewell, 1991). This also necessitates retraining of programmers when moving from one PLC vendor to another, or even from one PLC vendor's model to another.

- Lack of Tool Integration. In planning the plant and manufacturing processes, manufacturing and process engineers use a particular set of software tools, such as generic computer-aided design (CAD) software to produce drawings coupled with text documents produced with office automation tools such as word processing. The programmers and plant floor electricians use proprietary RLL programming language environments to program and maintain the PLC software. They may use additional support software such as spreadsheet or database software to keep track of the assignment of addressing information (the data table) in the PLC. The lack of integration of the tools used by these groups of people further contributes to complexity of system development and the difficulty of system maintenance. To quantify the costs involved, it has been projected that the development and integration costs of the various software packages in an automated factory will account for seventy percent of the total capital investment in future manufacturing systems (Ayres, 1989).

In summary, the development of real-time control software, and in particular, the development of control software for manufacturing using programmable logic controllers is problematic. The current suite of tools, such as generic drawing and office automation tools for specification and vendor specific RLL editors for programming, are not integrated. This leads to poor coupling between the design stage and the programming, testing and maintenance phases, requiring manual translation of

design into code which then results in the lack of design maintenance. Furthermore, the current tools do not supply domain-specific assistance nor do they control writing the specifications to a standard and do not support analysis and inspection of the design. These problems will be further elaborated in Chapter 4 where they were identified at the industry partner in this investigation.

1.4 Summary

Computer-based control is used as a means of automating the necessary schemes for control of manufacturing devices in the production process. The role and cost of software in modern manufacturing has been recognised as a major consideration in automation of manufacturing processes.

The predominate control device in automated factories is the programmable logic controller. Current practice in the development of control software for PLCs reveals numerous problems, including increased complexity, lack of design maintenance, a lack of standardisation, problems in software portability, and a lack of software tool integration. Additionally, researchers recognise the development of real-time control software as posing some of the greatest challenges in system development.

CASE tools present the promise of addressing problems in the software life cycle for developing manufacturing control software. Investigators have found that CASE tools have the potential to improve the production of software-related artefacts, to improve coordination between individuals involved in the development process, and result in overall organisational improvements, but that the impact of CASE tool effectiveness in industry is inconclusive (Henderson, 1990; Brown, 1994; Norman, 1989; Lempp, 1988; Card, 1987). Moreover, research and experience with CASE tools to date shows surprisingly little use of CASE tools at the requirements specification stage, but the almost universal use of generic office-automation tools (Potts, 1993).

The aim of this research is to investigate the support that appropriate CASE tools could have on the software development life cycle for control software for PLCs. As will be reviewed in Chapter 3, research in CASE tools suggest that:

- CASE tools that incorporate domain-specific methods are more likely to be adopted and be effective than are generic tools (Potts, 1993) and that domain-specific tools may help an individual user to understand and use the tools effectively (Kumar, 1992; Henderson, 1990; Davis, 1989; McClure, 1988).
- Design maintenance will be encouraged by CASE tools that support automatic generation of the software from the specification. Automatic code generation that is domain specific is more likely to be effective than generic code generation tools (Baxter, 1992; Barstow, 1991; Rich, 1988; Barstow, 1985).
- CASE tools should incorporate model-based analysis of specification to permit analysis and inspection of the design prior to code generation (Kopetz, 1991; Harel, 1992).
- CASE tools have the potential to improve individual and organisational effectiveness through improvements in the areas of production, coordination, and organisational technology (Henderson, 1990).

The major proposition of this investigation is that CASE tools based on the above theory will be adopted and will be effective in improving practice in the development of PLC control software. The research approach used herein is to design and validate a CASE tool targeted toward the domain of batch manufacturing control that incorporates features outlined above and then to assess its potential impact in the areas of production, coordination, and organisational support. Both the design and assessment are carried out in cooperation with industrial practitioners. Chapter 2 addresses the research approach in detail.

1.5 Outline of the Dissertation

Chapter 2 reviews research approaches that have been used to investigate CASE tools and identifies the approach used in this investigation.

Chapter 3 is used to identify a framework of ideas for development of a CASE tool to support the development of PLC control software. Areas surveyed include: a

definition of domain specific methods, a guide for developing application generators, CASE tools and a generic tool architecture, models and abstractions used for specification of real-time control software and associated CASE tools, programming languages used for PLC software, past work in the development of higher-level PLC specification languages and code generators, a CASE tool architecture for PLC software development, a review of general approaches to automatic programming, the concept of model-based analysis, and the potential areas where CASE tools may impact an organisation.

Chapter 4 reviews the problem domain and introduces the industry partner. Batch process control is defined and a representative structure and command language for specification of batch control is identified from the literature. The representative model is compared to that used at the industry partner and the two are found to be quite similar. The software development life cycle at the industry partner is reviewed, along with ladder logic coding conventions. The chapter concludes with an analysis of the software tools utilised in the current life cycle.

Chapter 5 describes the initial intervention into the organisation and the definition of the roles of the researcher and the industrial partner. This initial intervention and investigation led to the articulation of an approach to guide the design of domain-specific CASE tools that is then used to produce a prototype CASE methodology companion.

Chapter 6 presents the architecture, user interface, and examples of the prototype methodology companion developed for the investigation.

Chapter 7 describes an assessment of the CASE tools. The assessment methodology is presented, which utilises structured interviews with practitioners in industry. The results of the interviews are summarised and are used to assess the effectiveness of the CASE tool and to appraise the validity of the underlying framework of ideas that are incorporated into the CASE tool.

Chapter 8 presents the summary and conclusion of the investigation. The findings are summarised, the research methodology is appraised, and directions for future research are identified.

A goal of this investigation is to determine whether industrial practice can be improved through the use of CASE technology. The next chapter serves to review research approaches that have been used in investigations of CASE technology and identifies an approach applicable to the aforementioned goal.

Chapter 2 Research Approach

This chapter motivates the need for applied research that is done in collaboration with industry. Research approaches identified as appropriate for work in information systems, software engineering, and CASE technology are reviewed and action research is identified as a method that fits the purposes of this investigation. The characteristics of action research are reviewed, and its application to this investigation is presented.

2.1 Research Approaches

In a critical examination of software engineering research strategies, Potts laments the failure of much of past research to influence industrial practice (Potts, 1993). He notes that most research in the past employed artificial setting research, which he labels the "research-then-transfer" strategy. Instead, he calls for more intervention-based natural setting research which he calls the "industry-as-laboratory" strategy. In this approach a real system-development project that exhibits the problems of interest becomes a way to obtain knowledge and appreciate its significance (Potts, 1993). These studies exhibit scale, complexity, unpredictability, evolution, and change that are typically missing from artificial setting research. Moreover, by being more realistic, this strategy maximises the probability that results will be transferable to industry and also provides a mechanism for evaluating the research.

Other researchers have identified the need for more of this kind of research. In a literature review of a large number of studies aimed at evaluating the effectiveness of information systems, DeLone, et. al, notes the need for more industry-oriented (field) research aimed at measuring the influence of the information systems on organisational performance (DeLone, 1992). Richard A. DeMillo, head of the Software Engineering Research Consortium and a professor at Purdue University, states that "we need to have places where we can gather data and try things out. The only way to do that is to have a real software development organisation as a partner" (Gibbs, 1994). Kyng states that computer system design needs to be organised as a cooperative activity among

users and designers, emphasising that the design process should be grounded in the practice of users and employ concepts and tools that the users can grasp based on resemblance with concepts and tools from the application area (Kyng, 1991).

Since this research is targeted toward solving problems in a specific domain, and since there is a demonstrated need for industry-based studies, a cooperative approach with industry (herein called the "industry-as-laboratory" approach) is an appropriate overall research strategy. To identify a theoretical base for this approach, a review of research strategies was conducted.

A number of taxonomies and reviews of information systems research approaches are available (Hamilton, 1992; Galliers, 1992; Jenkins, 1985; Benbasat, 1984). A review of methods that have been applied to CASE research is found in (Wynekoop, 1991). This review identified eleven strategies, classified as either natural, artificial, environment-independent, or normative. The classifications and their constituent strategies are:

- Natural Setting Research. Methods conducted in actual organisations. Strategies include case studies, field studies, field experiments and action research.
- Artificial Setting Research. Methods where the environment is established by the researcher. Strategies include laboratory experiments or simulations.
- Environment Independent Research. Methods that assume that the process studied, the participants' responses, and the product of the research are not affected by the setting. Strategies include sample surveys, basic research, and applied research methods.
- Normative Writings. Work aimed at concept development to organise ideas or present new perspectives to stimulate research and action.

The industry-as-laboratory approach fits the natural setting category of approaches. These methods include: Case studies, defined as intensive evaluations of small samples of entities such as groups, organisations, individuals, systems, or tools; Field studies, which are used as an ex post facto method of evaluating changes in organisational systems based on recall and self-report of participants; Field experiments, which are characterised by the manipulation of one or more independent variables to observe the

effect on dependent variables, while controlling other extraneous variables; and action research which is a strategy where the researcher participates in an intervention in an organisation or activity to be studied while simultaneously evaluating the results (Wynekoop, 1991). Another definition of action research is *applied research where there is an attempt to obtain results of practical value to groups with whom the researcher is allied, while at the same time adding to theoretical knowledge* (Galliers, 1992).

Wynekoop makes the further observations of action research. Having the researcher as a participant provides a rich way of applying theory to practice and evaluating its worth. Moreover, although in their literature review the authors found no examples of action research to study CASE tools, Wynekoop suggests that action research would provide useful insights on how and why different features of a CASE product are used or not used.

Since action research provides a research design fitting to the industry-as-laboratory approach, and action research has been recommended as an approach for the study of CASE tools, and because this research aims ultimately to improve practice in industry, it was adopted as the research methodology. The following section presents a more detailed review of action research.

2.2 Action Research

The following is a definition of action research from (Benbasat, 1987):

> These (action research) are studies in which the author, usually a researcher, is a participant in the implementation of a system, but simultaneously wants to evaluate a certain intervention technique. Action research is conducted by those whose original intent is to conduct research while effecting change. The action researcher is not an independent observer, but becomes a participant, and the process of change becomes the subject of research. Thus, the researcher has two objectives: to take action to solve a problem and to contribute to a set of system development concepts. Although action research is similar in many respects to case study research, case study research is targeted toward efforts where research questions are specified prior to the study by researchers who are observers/investigators rather than participants.

Characteristics of action research (AR) that make it an appropriate approach for this research are the following (Susman, 1978). Action research is:

- Future oriented. Action research is oriented toward dealing with the practical concerns of people.
- Collaborative. Interdependence between the researcher and the client system is an essential feature of action research, and the direction of the research process will be partly a function of the needs and competencies of the two.
- Implies System Development. The aim of action research is to build appropriate structures, to build the necessary system and competencies, and to modify the relationship of the system to its relevant environment. The focus is on generating the necessary communication and problem-solving procedures. *The infrastructure of the system, which the action research generates, is the key instrument for alleviating the immediate problematic situation, and generating new knowledge about the system.*
- Grounded in Action. In action research, theory provides a guide for what should be considered in the diagnosis of problems, as well as for generating possible courses of action to deal with the problems. Action research *contributes to the development of theory by taking actions guided by theory and evaluating their consequences* for the problems members of the organisation face.
- Agnostic. The action researcher recognises that his or her theories and prescriptions for action are subject to re-examination and reformation upon entering new research situations. The objectives, the problem, and the method of the research must be generated from the process itself, and the consequences of selected actions cannot be fully known ahead of time.
- Situational. Appropriate action is based not on knowledge of the replications of previously observed relationships between actions and outcomes, but is based on knowing how particular actors define their present situations or on achieving consensus on defining situations so that actions will produce intended outcomes.

Although action research resembles consulting, it is by the interpretive or critical perspective that action research can be seen to generate knowledge to be used by the sponsoring organisation and the researchers. Differences between action research and consultancy include: the interpretive act and the public reporting of the project upon its conclusion (Jonsson, 1991), and the process of critical inquiry and the deliberate process of reflective learning (Checkland, 1991).

Examples of applications of the method to information systems research can be found in (Wood-Harper, 1985; Antill, 1985; Checkland, 1990). Applications in other fields are described in (Foster, 1972; Gibson, 1975; Chisholm, 1993; Elden, 1993).

Checkland describes the action research methodology as follows (Checkland, 1991).

> Initially the researcher will select a real-world situation as being potentially relevant to research themes significant to him or her. Then it is important and prudent carefully to negotiate the respective roles of the researcher and the people in the problem situation. Next, from the research point of view...it is essential to declare the framework of ideas and the methodology in which they are embodied. Substantive work can now begin, consisting of the involvement of the researcher in the unfolding situation with a view to helping bring about change deemed 'improvements'. While doing this the researcher tries to make sense of the accumulating experience, doing so by means of the declared framework and methodology. This may cause a rethinking of earlier stages...Finally the researcher exits from the situation (which is essentially an arbitrary act, since human situations continue to evolve through time) and reviews the experience in order to extract the various kinds of lesson.

Checkland emphasises the importance of identifying the explicit methodological framework, declared in advance by the action researcher, as a means of making the findings coherent, justifying them, and making them potentially transferable.

Checkland summarises the action research process as follows:

1. Enter the problem situation
2. Establish roles
3. Declare the methodological framework
4. Take part in the change process
5. Rethink 2, 3, and 4
6. Exit
7. Reflect and record learning in relation to the framework of ideas, the methodology, and the action situation.

The research approach is often described as cyclic in nature, as is shown by step 5 above, which iterates through steps 2, 3, and 4. Susman also describes action research as a cyclic process of diagnosing, action planning, action taking, evaluating, and learning. Figure 2.1 summarises Susman's model of action research.

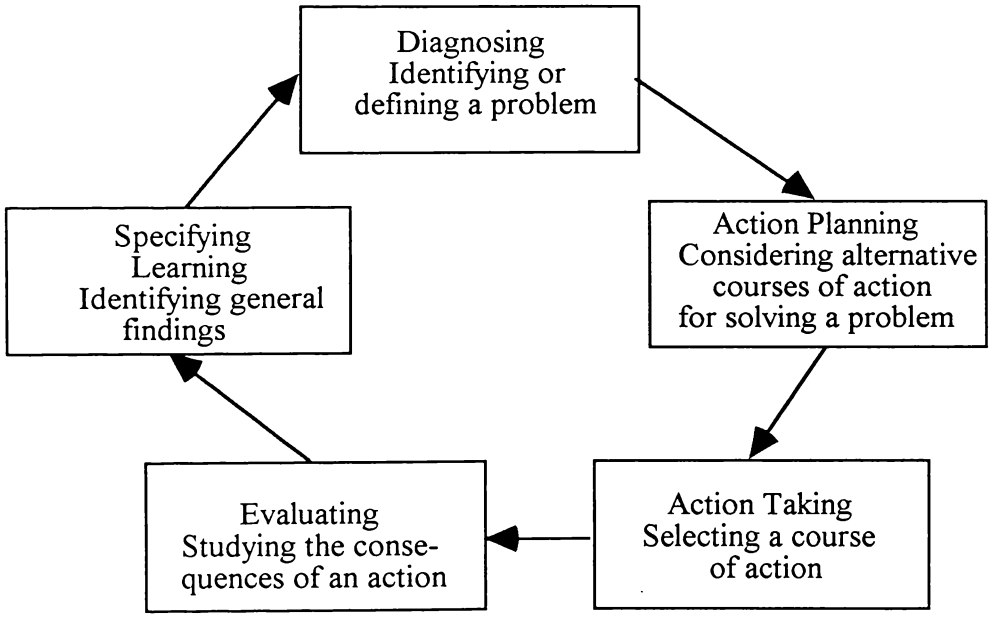


Figure 2.1. Cyclical Process of Research Approach. (Susman, 1978)

The research approach involves the intervention of the researcher into a client organisation. The next section reviews possible points of intervention.

2.3 Locus of Intervention

In a review of applied approaches to facilitate change in organisations, Leavitt suggests that industrial organisations can be viewed as complex systems in which at

least four interacting variables loom especially large; *task* variables, *structural* variables, *technological* variables, and *human* variables (Leavitt, 1965). The task refers to the organisation's reason for existence: the production of goods and services. The structure means systems of communication, authority, and work flow. The technology refers to direct problem-solving inventions like machinery or computers. The human variable refers to the people in the organisation. These four variables are highly interdependent so that change in any one usually results in a change in others. Figure 2.2 illustrates these four interacting variables.

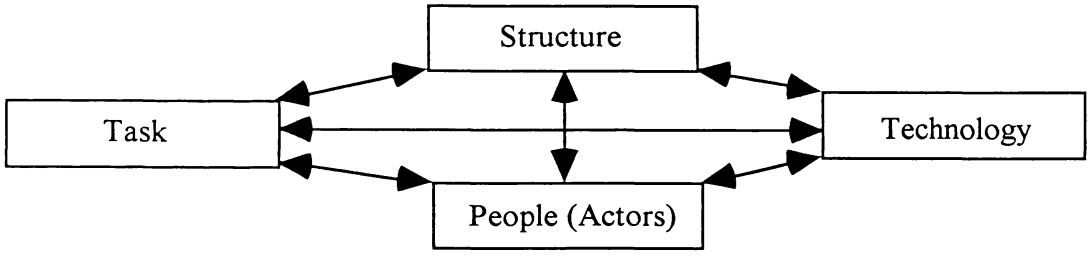


Figure 2.2. Interacting Variables in an Organisation (Leavitt, 1965).

Usually efforts to effect change are designed ultimately to influence the task variable by modifying either the structure, the technology, or the people. Leavitt states "Given, for example, the same statement of a problem by a client firm, one group of practitioners will try to solve it first by modifying structure ("What you need to do is decentralise"), another by changing technology ("Let's computerise"), or a third by changing people ("We'll set up a sensitivity training program")" (Leavitt, 1965). Thus, either the structure, technology, or people could serve as a preferred point of entry by the action researcher for bringing about change.

Most efforts to bring about change, whether they begin with people, technology, structure, or task must deal with the others (Leavitt, 1965). Thus, although the focus of this research is upon influencing the task of developing PLC software through the introduction of CASE technology, an aim of the investigation is to identify possible changes in tasks performed, structural changes, and changes in people's roles and activities that may result from the introduction of the CASE technology.

An example of such changes is described by Orlikowski in an empirical study of organisational changes triggered by the introduction of CASE tools into a large software development consulting firm (Orlikowski, 1989). One finding was that social relations on project teams can be disrupted and changed as a consequence of the deployment of CASE tools. The study reports behaviour including territorialism, resentment, and even rebellion among team members who felt that their job functions were threatened by the imposition of CASE technology. In this organisation, CASE technology was mandated as company policy and its implementation directed by teams of technical specialists. The main source of disruption found in this study appears to be due to the extreme specialisation of roles that evolved in the organisation because the users of the CASE technology (analysts and programmers) were given little control over the development and implementation of the CASE tools, which was the domain of the technical specialists. As a result, the users felt a loss of power and autonomy, and at the same time resented the growing power of the technical specialists.

Thus, an aim of this investigation is to study the potential influence on the structure and people of the organisation that results from the introduction of CASE technology.

2.4 Software Engineering Principles

Basili presents a number of recommendations that can provide a basis for sound software engineering theory and practice that fit closely the action research approach (Basili, 1988). Basili states the following implications of these principles: We need to *better understand* the software construction process and product; Such an understanding will allow us to *plan what we need to do* and improve over current practices; To make plans operational, we need to *specify how we are going to effect* the construction processes and their analysis; The *execution of these plans involves the construction of products* and analysis of the constructive processes and resulting products; All of these implications need to be integrated in such a way that they *allow for learning* and feedback so that we can improve the software development processes and products.

Basili's principles fit precisely the action research strategy. Table 2.1 summarises this correspondence.

Susman's Action Research Model	Basili's Software Engineering Principles
Diagnosing: Identifying or defining a problem	Characterise the current project environment.
Action Planning: Considering alternative courses of action for solving a problem	Plan what to do to improve over our current practices. State precise goals and quantify them. Choose an appropriate life cycle model, methods, and tools to achieve the goals.
Action Taking: Selecting and carrying out a course of action	Make those plans operational. Execute the model and construct the products.
Evaluating: Studying the consequences of an action	Collect the prescribed data and validate it. Analyse the data to evaluate current practice.
Specify Learning: Identifying general findings	Determine problems, record findings, and make recommendations for improvement.

Table 2.1. Basili's Improvement Oriented Paradigm.

Basili also presents a paradigm aimed at providing a mechanism for formalising the tasks shown in Table 2.1 called the goal/question/metric (GQM) paradigm. It requires the definition of goals, their refinement into quantifiable product-related and process-related questions (sub-goals) that will lead to identification of appropriate measurement metrics, which will supply answers to the questions (Grady, 1987).

2.5 Assessment of Findings of Action Research

Researchers who advocate the use of applied field-oriented research approaches such as action research have developed methods to assesss findings that may result from these approaches. Argyris states that research findings, or knowledge, should include empirically disconfirmable propositions that can be organised into theory and falsified by practitioners in real-life situations (Argyris, 1985). This requirement asks that propositions be characterised by features that allow practitioners to disconfirm them. Thus propositions are made public, based on observable data, and conditions should be provided that are conducive to testing them. Yin confirms this approach by noting that the first and preferred strategy for analysing case study-like

evidence is to follow the theoretical propositions that led to the study (Yin, 1989). It is also consistent with Basili's approach which calls for the statement of goals, which can then be operationalised in the form of questions, which will lead to measures that can be used to answer the questions (Basili, 1988).

The approach used in this investigation is based upon the recommendations outlined above. Expected outcomes resulting from the introduction of CASE technology are identified based upon a framework of recommendations identified in the literature. A prototype CASE tool that incorporates the framework of recommendations is introduced to practitioners, and their informed judgement is used to test the propositions. This information is collected through structured interviews which are recorded, transcribed, indexed, and categorised.

The informed judgement of practitioners is utilised for the assessment since this investigation is a natural-setting form of research. Informed judgement is recommended (Weiderman, 1986), as opposed to controlled experimentation, which is an inappropriate method for natural-setting research. Moreover, Wood and Wood report that to try to answer questions such as *which methods and tools are best* using controlled experimentation are not within the grasp of existing resources or technology (Wood, 1989).

2.6 Application of the Research Approach

Referring back to Section 2.2, Checkland's paradigm for action research calls for declaration of a research theme which includes a theoretical framework and a methodology for its application to a real world problem situation. In this investigation, the research theme is the investigation of the potential for CASE technology and its effectiveness in improving the development of batch control software for PLCs. The theoretical framework consists of claims from the literature concerning the CASE tools including the potential of domain-specific methods, design maintenance and automatic code generation, model-based analysis, and past work in developing tools to support construction of PLC software. A methodology to guide the development of domain-

specific CASE technology is also identified. The next step in the approach, action taking, is to collaborate with the client organisation to change the technology used to develop control software. The result of this collaboration is the design of a domain-specific CASE tool, based upon the theoretical framework, and the evaluation of the tool in the target domain. The action step serves to generate information to assess the effectiveness of the CASE tools, the underlying theoretical framework and methodology upon which they are based, and the influence of the CASE tools on industrial practice.

Checkland makes the following point about this research approach (Checkland, 1991). The researcher immerses himself or herself in a human problem situation and seeks learning out of the achievement of practical ends. Such work has no option but to follow the situation along whatever path it takes as it unfolds through time. Thus, the researcher cannot give in advance a precise account of the intended substantive content of the research. It is the cyclic nature of the methodology, where findings lead to new actions which in turn lead to additional findings, that accounts for the inability of the research to precisely state, at the outset of the investigation, all aspects of the intended research.

2.7 Researcher's Preconceptions and Interests

Since the action research methodology involves the intervention of the researcher into the problem situation, it is important that the researcher make clear his or her preconceptions and interests in the field of study in order that the reader of the research may judge for him or herself whether the conclusions of the research are sufficiently objective.

The researcher has had a long-standing interest in the use of software tools and environments to support software development, and has also had prior interest in programming languages to support programming for automated manufacturing, resulting in the belief that improvements in such languages are needed (Troy, 1992). It is the belief of the researcher that new and appropriate tools are a necessary ingredient

for improving practice in the development of software. For example, the researcher believes that the advancements in programming languages over the years, from assembly language to high level languages, represent the advancement of programming tools without which the practice of software development would have shown little improvement over time.

However, the researcher is skeptical as to the effectiveness of generic CASE tools. In an investigation of early CASE tools the author concluded that most generic CASE tools did not fit the needs of the organisation for which the study was conducted (Troy, 1987).

Thus, the researcher has a bias in favour of software development tools, he believes that appropriate languages are necessary to support programming for manufacturing, and he holds skepticism about the effectiveness of generic CASE tools.

2.8 Summary

Researchers in software engineering have identified the need to perform more natural-setting research, dubbed the industry-as-laboratory approach. The action research approach is identified as a method to guide this form of research, and its application in this investigation described. The point of intervention in the study is the technology utilised in the development of control code. In particular, the study aims to introduce CASE technology and then to assess the influence of this change on the task of developing control software and also upon the people and the structure of the organisation.

The action research approach calls for identification of an explicit framework of ideas, a methodology to apply the framework, the development of an infrastructure based upon the methodology and framework and its application in practice, and then reflection and interpretation of the results of the action taken. The next chapter presents a review of the framework of past work in CASE tools and PLC software development that forms the theoretical foundation for the CASE tool developed for this investigation.

Chapter 3 Literature Review

This chapter serves to build a framework of ideas upon which to design a CASE tool to support the domain of manufacturing control programming. Examples and types of CASE tools are presented, along with a generic architecture for a CASE tool. Recommendations from the literature concerning important features of CASE tools (model-based analysis, design maintenance, and automatic code generation) are presented. Past work in CASE technology and its application to the domains of real time control and PLC programming are reviewed. Also, a conceptual model for defining the concept of domain-specificity is described.

3.1 CASE Environments and Tools

As stated in Chapter 1, CASE is a technology that involves the use of computer-based tools to provide automated support for the software development process (Brown, 1994; Fuggetta, 1993; Henderson, 1990; McClure, 1989). The roots of CASE technology are reviewed in (McDermid, 1991). Initially individual software tools were built to assist with particular software development tasks, the goal of such tools being to automate the mundane aspects of the software development through the use of computer support. Examples of these sort of software tools include editors (text and graphical), programming tools (eg., assemblers, compilers, debuggers), verification and validation tools (eg., flowcharters, syntax checkers, dynamic analysers, test case generators), configuration management (eg., version control, library managers), metrics and measurement tools (eg., code and execution analysers), project management (eg., project planning, cost estimation, project notebooks), and miscellaneous tools (Fuggetta, 1993).

As software tools proliferated it became apparent that many incompatible tools were being developed and it was advocated that these tools be brought together into an integrated environment (McDermid, 1991). In the business realm the notion of a data dictionary was exploited as the corporate data base of the computing department

around which a number of support tools were provided to support the software development life cycle (DeMarco, 1978). In more technical-oriented computing an integrated programming environment to support Ada software development, known as the Stoneman report, was proposed in 1980 (Buxton, 1980). These developments led to the idea of a software development environment [1] (SDE) as a collection of software tools and other components together with a life cycle model and an integration approach that supports most or all of the interactions that occur among the environment components, and between the users of the environment and the environment itself (Brown, 1994; Basili, 1988). The goal of a SDE is to augment or automate all of (or as much as possible) the activities comprising the software development process (Dart, 1987; Basili, 1988).

In the fifteen years since the Stoneman report, Brown states that it has been argued that no commercially successful full life cycle SDE has been developed (Brown, 1994). Reasons cited for this lack of success are that SDE technology has proven to be far more complex than initially anticipated and the rapid changes in underlying technologies. Current trends are toward development of limited collections of CASE tool solutions, often using combinations of commercially available tools.

CASE tools can be grouped into three basic categories (McClure, 1989):

- CASE toolkits: sets of integrated tools that support one type of software development function or job class
- CASE workbenches: general-purpose software development environments supporting the full range of the software job
- CASE methodology companions: computerised assistance for a particular software development methodology.

---

[1] Software development environments are also known as CASE environments, software engineering environments (SEE), or integrated project support environments (IPSE) [Brown, 1994].

CASE toolkits consist of tools that support one particular phase of the software life cycle. For example, there are analyst, designer, programmer, and maintainer toolkits. A CASE workbench is an assembly of integrated tools whose function is to automate all phases (or as much as possible) of the life cycle as well as project management activities. A CASE methodology companion can be either a toolkit or a workbench that structures the software development process according to the steps and rules of a particular structured methodology. Information about the methodology is embedded into the methodology companion in order to facilitate following the steps and rules in a development methodology and enforce standards (McClure, 1988). Methodology companions provide information about the methodology in the form of help, menus, and functions that control what a developer can do and provide feedback on the use of the methodology (Vessey, 1992).

The promise of CASE is that automated support for some aspects of the software development life cycle will:

- increase productivity and reduce the cost of software development
- improve the quality (eg., reliability, useability, performance) of software products
- keep documentation in step with software products as they evolve
- facilitate maintenance of existing software systems
- make the software engineer's task less odious and more enjoyable (Brown, 1994, McClure, 1989).

However, as was pointed out in Section 1.2, research is inconclusive as to the benefits of and acceptance of CASE technology.

3.2 Domain Specific Methods and CASE

In Chapter 1 it was noted that CASE tools have found limited use in the specification phase of the software development life cycle, and also that the impact of CASE tools in industrial practice is unclear (Brown, 1994, Potts, 1993, Henderson,

1990, Norman, 1989; Lempp, 1988; Card, 1987). Theories to explain these limitations include:

- The lack of acceptance is because tools based upon generic, domain-independent languages are unsuccessful in capturing user requirements (Potts, 1993)
- People will use technology to the extent that they believe it will help them perform their jobs better (Davis, 1989; Schultz, 1975)
- CASE tools should act as methodology companions by incorporating information about a particular software development methodology (McClure, 1988)
- Software development environments should be tailorable so that they can be altered or adapted to suit a particular project environment (personnel, type of application, etc.) and organisation (Basili, 1988).
- The effectiveness of software tools is related to the intellectual effort required to use them (Krueger, 1992)

The above results suggest that CASE tools should be domain specific by incorporating support for a domain dependent methodology, language, and knowledge. Furthermore, there is no design methodology that is best in all situations. A methodology should be chosen contingent on variables within and around the problem situation (Kumar, 1992). A conceptual basis for analysing domain-specificity will now be developed.

Software engineering technologies can be analysed using the dimensions of *abstraction* and *specialisation* (Krueger, 1992; Keebler, 1993). Abstraction refers to the concept of identifying essential properties while simultaneously eliminating non essential properties (Marciniak, 1994). Abstraction is the primary mechanism to manage complexity and control details. Specialisation is the process of taking a generic software artefact and specialising it through parameters, transformations, constraints, or some other form of refinement (Krueger, 1992).

Krueger elaborates on the use of abstraction as follows (Krueger, 1992). Every software abstraction has two levels: the *abstraction specification* and the *abstraction realisation* (or implementation). For example, in a compiler the abstraction specification is the syntax of the source code and the abstraction realisation is the compiled machine code. The abstraction specification typically describes what the abstraction does, whereas the realisation describes how it is done.

An abstraction also has three parts: a hidden part, a variable part, and a fixed part. The hidden part consists of the details in the realisation that are not visible in the specification. The variable and fixed parts are visible in the specification. The variable part can be changed by the user whereas the fixed part is invariant. In essence, the variable part allows the user to specialise the tool and thus leads to a collection of alternate realisations.

The creator of a software tool decides what information will be useful to the users of tool and puts this information into the abstraction specification. The parts that the user might want to change are placed in the variable part. Information that must be visible to the user but cannot be changed is placed in the fixed part.

As an aid to evaluating the effort required to use a tool, Krueger introduces the intuitive concept of *cognitive distance* (Krueger, 1992). Cognitive distance is defined as the amount of intellectual effort that must be expended by software developers in order to take a software system from one stage of development to another. For the creator of a software tool, the goal is to minimise cognitive distance by (1) using fixed and variable abstractions that are both succinct and expressive, and (2) maximising the fixed and hidden parts of the abstractions, and (3) using automated translation from the abstraction specification to the realisation.

The incorporation of domain specific methods into a CASE tool minimises the cognitive distance of the tool in several ways. The following are adapted from (Krueger, 1992):

- Capitalising on the user's understanding of the application domain rather than requiring users to derive domain semantics repeatedly from first principles as in conventional generic CASE tool technology.
- Capturing standards in the fixed part of the CASE tool abstraction. This is the essence of a methodology companion.
- Use of a specification language that is succinct and expressive for the particular application domain.
- Use of automatic code generation to map the higher level specification to a lower level realisation appropriate for the particular domain.

A domain specific CASE tool based upon the above guidelines bears resemblance to an *application generator* (Cleaveland, 1988). Application generators operate like programming language compilers but differ in that the input specifications are typically very high-level, special-purpose abstractions from a very narrow application domain (Krueger, 1992). The abstraction presented to the user of an application generator typically comes directly from the corresponding application domain because different applications need different models of data and computation (Cleaveland, 1988). The range of the variable and hidden part for an application generator determines the range of systems that can be generated. This is referred to as the *domain width* (or coverage) of the generator (Cleaveland, 1988). Most application generators have a narrow domain width. Examples of application generators include systems to create user interfaces, switching software, test drivers, hardware interprocess communication, CAD design process descriptions, and printed circuit layout generators.

Cleaveland describes the processing of building an application generator as follows (Cleaveland, 1988):

- Recognising an appropriate domain. Domains that are amenable to generator technology will have implementations with recognisable patterns at the source code level, have natural control and data abstractions, contain automatable transformations, or have dispersed information that can be consolidated.

- Defining domain boundaries. This step involves deciding what aspects of the domain should be included in the generator. Increasing the domain coverage allows the generator to handle more problems but typically makes the generator less efficient and harder to use.
- Defining and underlying model. The 80/20 rule, where everything in a domain does not fall into a clean computational model, sometimes applies (Levy, 1986). In this case provision must be made so that the 20% outside the model can be programmed directly into the implementation technology.
- Deciding the variant and invariant parts. Deciding what aspects of the model are under the control of the user (the variable part) and what aspects are fixed or hidden (the invariant parts).
- Defining the specification input. Options include textual specification languages, templates, graphical diagrams, interactive menu-driven dialogue, and structure-oriented editing.
- Defining products. Products of a generator could be a program, documentation, test data, or other products.
- Implementing the generator.

Domain analysis is the process of identifying and organising knowledge about a problem domain to support the description and solution of problems in the domain (Arango, 1991). After organising the knowledge about a problem domain, the necessary infrastructure for an domain specific tool can be designed.

Sources of domain knowledge include technical literature, existing implementations, user surveys, expert advice, and current and future requirements. Outputs include domain languages, standards such as interfaces, and functional models such as generic architectures (Arango, 1991). Domain analysis is also used to produce domain languages which may be implemented by software components and transformed into programs, by, for example, automatic programming tools.

The next few sections in this chapter focus on a review of CASE tool architectures.

3.3 A Generic CASE Architecture

CASE tools typically consist of a repository of design information together with a set of tools for man-machine communication, generation and recording of project documentation, consistency checking, code generation, prototyping and simulation, report generation, query facilities, and possibly other functions (Sommerville, 1989; Halang, 1992). Figure 3.1 presents a schematic for a generic CASE tool architecture.

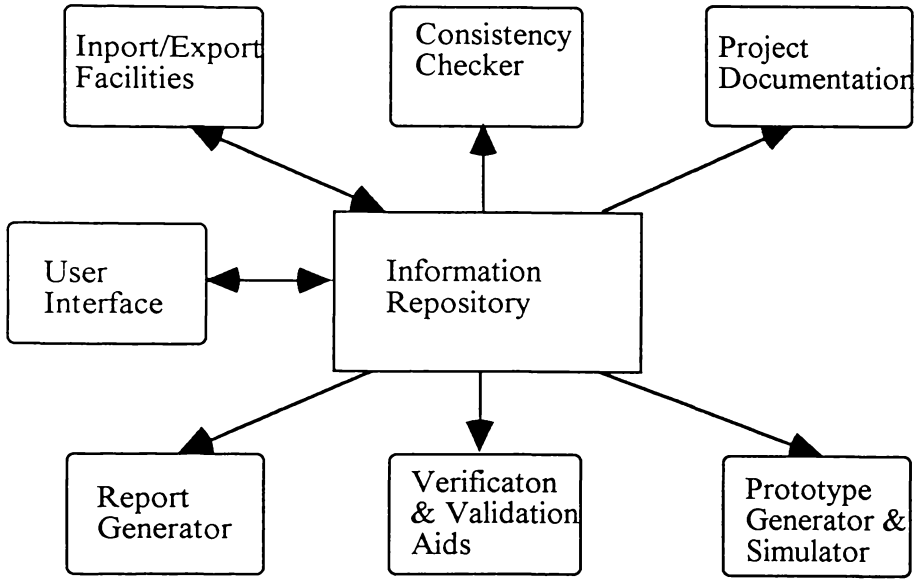


Figure 3.1. Generic CASE System Architecture.

The information repository is a data base of design information that is shared by the other modules of the CASE tool system. Information in the repository could include various kinds of design artefacts such as a data dictionary and specifications of a software system.

CASE tools are typically built to support a particular software development methodology, and the user interface then employs the specification methods of that methodology. For example, many commercial CASE tools support the structured analysis and design methodology based on DeMarco's methods (DeMarco, 1978). The specification methods employed by structured design are data flow diagrams and

structure charts, so a user interface for a CASE tool using this methodology would support drawing of these diagrams.

The consistency checker and verification and validation aids allow the user to analyse design documentation for completeness and consistency.

The project documentation module could support scheduling and estimation.

The report generator is used for generating information from the repository and other hard-copy project documentation.

The prototype generator and simulator module may include automatic code generation of a prototype system or simulation of the design prior to code generation.

Finally, the import/export facilities support addition or extraction of information to or from the repository and other software tools external to the CASE tool system.

Methodologies to support the development of real-time software are available in the literature. The following section examines representative real-time methods and CASE tools that employ those methods.

3.4 Generic Models for Real-Time Control

Most methods to support the design of real-time control systems employ the following three models, or views, of a system under development (Gee, 1992; Wood, 1989):

- the *functional* model, showing the relationship between the functions carried out by a system on the stored data.
- the *structural* (or *data*) model which deals with both system architecture and with stored data modelling.
- the *behavioural* model, depicting the response of the system to events, as influenced by the passage of time and the history of previous events.

For illustrative purposes, two examples of these models are presented. The first example is the real-time methodology of Hatley and Pirbhai, who suggest the use of a

process model, control model, and an architectural model (Hatley, 1987). The process model uses data flow diagrams to depict functional requirements of the system. It partitions the requirements into component functions, or processes, connected by data flows. The control model uses finite state diagrams to depict behavioural requirements of the system. The architecture model specifies the physical entities that make up the system, the information flow between the entities, and channels on which the information flows. (For certain circumstances, Hatley and Pirbhai also propose an Information Model to describe the structure of stored data, modelled using entity-relationship diagrams.) The three major models are illustrated in Figure 3.2.

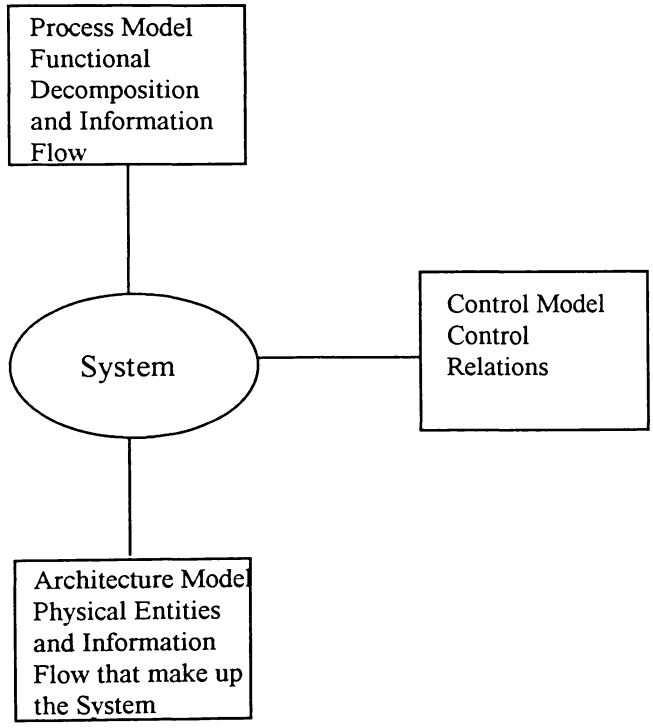


Figure 3.2. Modelling Methods Suggested by Hatley and Pirbhai (Hatley, 1987).

The second example is that of the STATEMATE system. STATEMATE is a commercial CASE tool that supports specification and analysis of a system from three related points of view: structural, functional, and behavioural (Harel, 1990). The structural view provides a hierarchical decomposition of a system into its physical components, called modules, and identifies the information that flows between them. Module charts are used to document the structural view. The functional view uses activity charts to identify a decomposition into activities and possible flow of

information, but it says little about how those activities and associated inputs and outputs are controlled. The behavioural view uses statecharts to specify control. These abstractions are summarised in Figure 3.3.

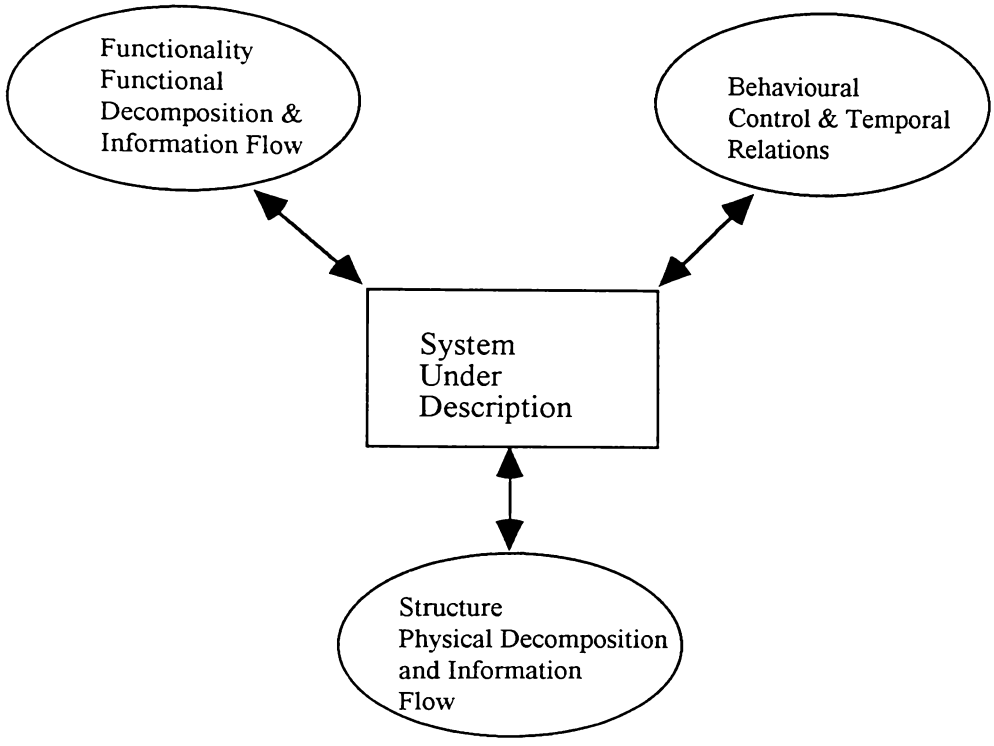


Figure 3.3. Modelling Methods in STATEMATE (Harel, 1990).

To summarise, most existing real-time modelling methods include aspects of structural, functional, and behavioural modelling. These models and their associated specification methods are examined further in the next section.

3.5 CASE Tools and Their Abstractions for Real-Time Control

CASE tools are available that support the development of real-time control software. Before reviewing CASE tools for control software development, some related families of tools are mentioned, primarily to note other classes of tools that are not addressed by this investigation. One family of tools, known as computer aided control engineering (CACE) tools, are those that are designed to assist with solving multi-variable control problems that are essentially numeric in nature. Surveys are available in (Munro, 1992; Herget, 1988; VanDerBoom, 1988; Jamshidi, 1985). Functions of these tools include transformation tools, eigenvalue/eigenvector

algorithms, minimal realisation techniques, describing function analysis, and curve fitting. Examples of these tools include $MATRIX_X$, and Control-C. These problems are not encountered in the domain of investigation in this research and were therefore not considered as potential CASE tool solutions. A survey of computer aided process planning (CAPP) systems is described in (Ahlgrim, 1989). These systems are designed to support the manufacturing function that establishes the machines, processes, and/or parameters that will be used to transform a piece part from its raw materials to its final form.

Returning back to the domain of real-time control, Wood and Wood, using a literature search, identified over 100 proposed software specification and design methods for real-time control systems (Wood, 1989). Next, through a questionnaire administered to affiliates of the Software Engineering Institute, they identified three methods as representing the predominate approaches used in practice. A total of 447 questionnaires were distributed. The three dominate methods are the techniques of Ward-Mellor (Ward, 1985; Ward, 1986), Hately-Pirbhai (Hatley, 1987), Harel (Harel, 1990). (Methods identified as similar to these three are Extended Systems Modelling Language (ESML) (Bruyn, 1988) which includes attributes of Hatley-Pirbhai and Ward-Mellor, and Systems Engineering Methodology (Wallace, 1987).)

The Ward-Mellor method is based on the earlier structured design method of Yourdon and DeMarco (DeMarco, 1978). The primary abstractions are data flow diagrams enhanced to show control processes, called event flows, as well as data flows. Behaviour is specified using finite state machines using state transition diagrams.

The Hately-Pibhai method is similar to Ward-Mellor in that it is derived from structure design. Enhanced data flow diagrams are also used, but separate diagrams are used for control flow and data flow and state-machines are used to specify control. A graphical template called an architectural context diagram (which can be decomposed) is used to model physical modules and the information flows between them (Hatley, 1987).

Harel's method utilises statecharts, activity charts, and structure charts. Statecharts are a modification to conventional state diagrams to support hierarchical decomposition (with statecharts, states are allowed to contain other states). Activity charts are similar to data flow diagrams, with control processes on the activity chart described by means of statecharts. Module charts are used to show physical components of the system.

To summarise, all of the above methods use data-flow concepts to model functional requirements (Halang, 1992; Wood, 1989). For the structural specification, Harel and Hatley-Pribhai provide their own graphical notation for architectural modelling while Ward-Mellor do not provide a distinct notation for architectural modelling (Wood, 1989). For modelling of data, Ward-Mellor specify the use of entity-relationship diagrams and Hatley-Pribhai recommend the use of the same; Harel does not provide for data modelling (Wood, 1989). For modelling behaviour, all of the methods use some form of finite state machine representation, but the methods differ in notation and emphasis. Harel uses the unique form of statecharts; Ward-Mellor emphasises conventional state-transition diagrams and tabular formats; Hatley-Pirbhai emphasises multiple tabular formats but also supports conventional state-transition diagrams and combined formats (Wood, 1989).

CASE tools are available that are based upon each of these approaches and surveys or descriptions are available in (Kavi, 1992; Ludwig, 1991; Chaar, 1991; Chaar, 1990). Representative examples are described below.

The STATEMATE system, introduced above in Section 3.4, is a set of graphically oriented CASE tools that implement Harel's statechart-based approach (Harel, 1990; Kavi, 1992). STATEMATE makes it possible to execute, debug, and analyse a system specified using Harel's statechart notation. Tool functions include: graphics and forms editors, a simulation package and language, report and plot generators, analysis report generators, and a code generator. The code generator can generate Ada or C code for a prototype realisation of the system.

CARDtools is a set of CASE tools for the development of real-time embedded systems (Ready, 1991; Kavi, 1992; Ludwig, 1991). Support for real-time specification is based on Ward-Mellor methods including state transition diagrams and control flow diagramming. The tools enable the user to analyse the design through simulation.

Software through Pictures (StP) is used for embedded and commercial systems development that also uses methods similar to Hatley-Pribhai and Ward-Mellor (Interactive, 1991; Kavi, 1992). A collection of editors for different structured design methodologies are provided. Those that directly support real-time specification include data flow, control specification, and state transition diagram editors. The system includes a code generator for C, Pascal, and Ada.

Teamwork/RT supports real-time development using similar tools to Software through Pictures. Specification methods include control and data flow diagrams, process specifications, and control specifications using decision tables, process activation tables, state transition diagrams and state/event matrices (Ludwig, 1991). A code generator with support for Ada development is also included in the Teamwork system.

A variety of other methods and tools are identified in a survey by Chaar (Chaar, 1990; Chaar, 1991). The abstractions for specification of control identified in this survey include Grafcet and Petri nets (David, 1992), Coloured Petri nets (Viswanadham, 1987), and extensions of Petri nets (Zhou, 1990). Applications of these methods to the domain of PLC software development are further examined in Section 3.6.

To conclude this section, methods and CASE tools to support the development of real-time software typically support three modelling abstractions: functional, structural, and behavioural. The abstraction specification methods employed are typically enhanced data-flow methods for the functional, special graphical notations for the module architecture, entity-relationship diagrams for the data architecture, and some form of finite state machines for the behavioural specification.

The remaining sections in this chapter present a review of methods and tools designed specifically to support the domain of developing PLC manufacturing control software for PLCs. It should be noted that programming languages and computer aided software development tools for PLCs are predominantly available in vendor-specific form (Halang, 1992). The following survey concentrates on standards and academic investigations.

3.6 Languages and Code Generators for Programming PLCs

In Chapter 1 it was noted that several languages exist for programming PLCs, with the predominate language being relay ladder logic (RLL) and that the international community has recognised the need to standardise the programming languages used for PLCs. The International Electrotechnical Commission (IEC), Subcomittee 65B (formerly SC65A/WG6), has proposed a standard, known as IEC Standard 1131 for programmable controllers, that defines a family of languages that can be transformed into one another (IEC, 1991; Halang, 1992). Two are text-based: Instruction List (IL) and Structured Text (ST), and three are graphical: Ladder Diagram (LD), Function Block Diagram (FBD), and Sequential Function Chart (SFC).

Instruction List languages constitute the lowest end of the family of standard PLC languages. They are essentially assembly languages and are thus machine and vendor specific. To make IL languages less machine and vendor specific, the standard provides a model language for a hypothetical processor (Halang, 1992).

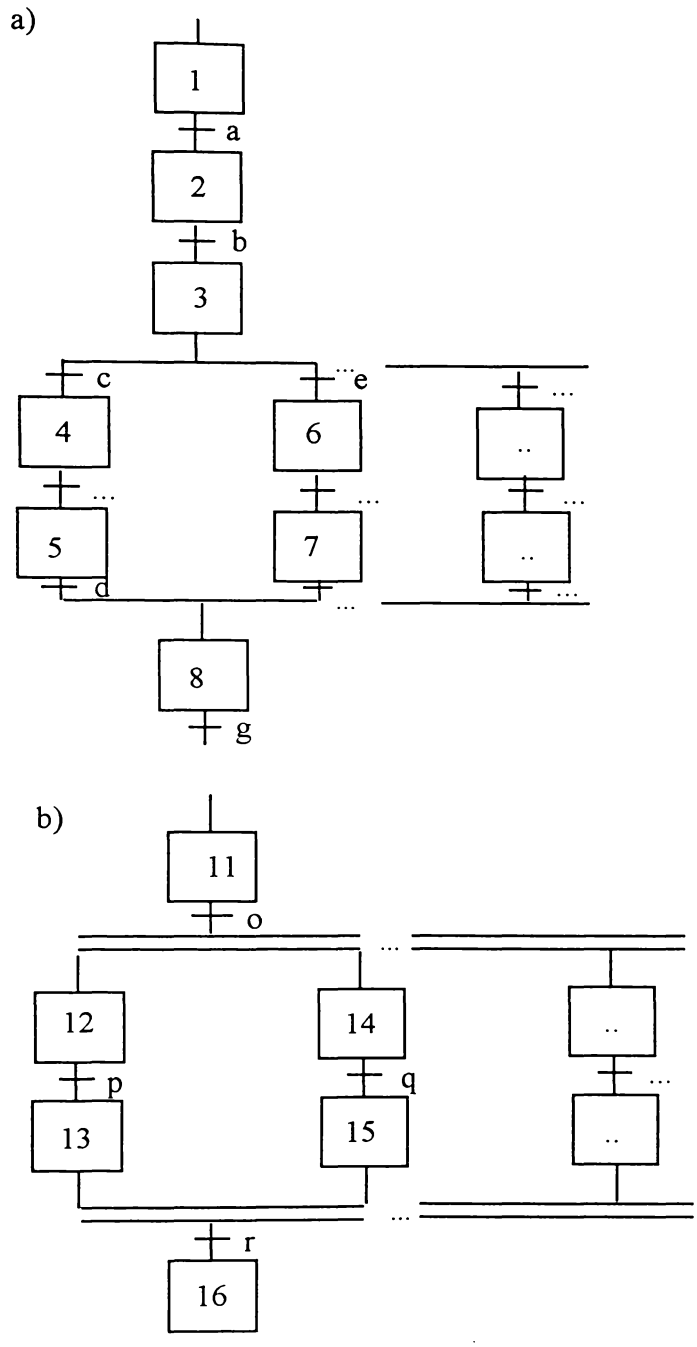
Structure Text is similar to third-generation high level programming languages, having a Pascal-like syntax and functionality and is therefore machine and vendor independent. Statements in the language include assignment, conditional, and loops.

Ladder Diagrams represent an attempt to standardise the relay ladder logic languages. To aid in standardisation, the IEC standard defines a unified syntax and basic language elements.

Function Block Diagram has been derived from blueprints of digital circuits, in which chips represent a certain module of functionality. The generalisation of this

concept leads to functions and function blocks that are depicted by rectangular boxes which may have inputs and outputs of various data types. Functions and function blocks may perform binary, numeric, analogue, or character processing and functions can be stored in libraries of function blocks. The graphic representation of logical and functional relationships are box symbols and connecting lines. Hierarchical decomposition is also supported. However, this form of representation turns out to be difficult to understand when complexity increases (Halang, 1992).

Sequential Function Chart (SFC), also known as Grafcet, was inspired by Petri nets with a few differences (David, 1992). Grafcet first appeared as a French standard, then as an international one in 1987 (IEC, 1988). A SFC diagram is composed of a sequence of steps and associated actions where steps are connected by transitions, and associated with each transition are conditions. Figure 3.4 illustrates the structure of sequential function charts.



(Note: ... represents possible expansion of the diagram)

Figure 3.4. Examples of Sequential Function Charts; (a) shows alternative branches, (b) shows parallel steps (Halang, 1992).

The steps are represented by boxes, which are linked to other steps by connectors and transitions. One or more actions can be associated with each step, and can be coded using any of the previous four IEC languages. The transitions are represented by the horizontal slashes on the connectors (lettered a, b, c, etc. in Figure 3.4). The

transition conditions are boolean expressions, typically formulated as Function Blocks or Structured Text (Halang, 1992).

At a given point in time during program execution, a step can be either active or inactive. When a step is active, its associated actions are enabled. As is true for Petri nets, a step remains active and causes the associated actions as long as the conditions of the subsequent transitions are not fulfilled.

Transitions are either enabled or not. A transition is considered enabled when all steps connected to the transition and directly preceding it are active. A transition "fires" when it is enabled and its transition condition has become true. The firing of a transition causes the simultaneous activation of the directly following step(s) and the deactivation of the immediately preceding ones (Halang, 1992).

Other methods for specifying PLC control have been proposed. Roberts has proposed the use of a graphical notation derived from the work of Hately-Pribhai and Ward-Mellor described above (Roberts, 1990). Languages for use by automatic PLC code generators have been proposed by several researchers. Ahmed has developed a limited-scope ladder logic generator for manufacturing systems that employ a number of pneumatic or hydraulic cylinders arranged to operate in a specific sequence (Ahmed, 1988). This system employs a command-line type user interface which allows the user to input the number of cylinders and the number of events and the sequencing of operations. The program will then generate an event sequence table and a ladder diagram. This system is narrow in scope, and does not address the domain under consideration in this project. Several researchers have developed PLC code generators that accept lists of operations and associated actions and conditions and generate the equivalent of ladder logic code. Bhatnagar and Linn have developed a PLC code generator that takes a list of operations with associated actions and enabling conditions (Bhatnagar, 1990). This list, presented to the system in text format, is converted to an internal format, represented as boolean logic, and stored in a database. The internal form can then be transformed into the machine code for a target PLC. Gunasena and

Lehtihet have developed a code generator that accepts a list of operations, "while" conditions (invariants), and "end" conditions (post-conditions), along with methods for specification of parallelism (Gunasena, 1991). The list of operations, given to the system in a textual format, can then be translated into the boolean equivalent of ladder logic code. Both of the systems described above are based upon textual specification languages. They lack mechanisms for hierarchical decomposition, thus limiting them to small applications. Further, the specification languages are not executable or simulated.

Several projects are reported by Krogh and his associates at Carnegie Mellon University. These projects utilise the approach described above, namely to specify the control system by coding a list of operations, enabling conditions, and actions and then transforming these operations into executable code. Pathak and Krogh report on a language, consisting of a text-based list of concurrent operations with associated enabling conditions and actions (Pathak, 1989). As the operations are developed, they are stored in a relational database, and when complete can then be transformed into C language code. In a related work, Krogh, Willson, and Pathak report on a system that utilises the same approach to the specification language (operation/condition/action list) but automatically extracts a Petri net model of the control and process logic (Krogh, 1988; Krogh, 1987a). Various Petri net-based analysis tools are provided to investigate the consistency of the specification, and the control logic can be translated into C code. Krogh and Ekberg report on a system wherein the user builds up a data base of operations, resources, actuators, and sensors (Krogh, 1987b; Ekberg, 1987). Operation records contain the logical conditions for initiating each operation, and the resulting states following the operation. Operation records point to related resource records and their associated states. Operation records also point to actuator records that identify the physical devices that execute the operation. Resource records point to sensor records that identify the sensors used to evaluate the state of a resource. The basis for the identification of the operations and resources is a modified Petri net model called operation/resource nets (O/R nets) (Beck, 1986).

Devanathan developed a PLC code generation system consisting of two programs. The first, called AILISTD, accepts a restricted English textual language description of a sequential control system and converts this into a state transition diagram (STD). The second program, called PROLOC, automatically translates the STD into relay ladder for implementation for an OMRON S6 PLC (Devanathan, 1991). An example of Devanathan's language, which he describes as employing a 'cause' and 'effect' structure, is as follows:

1. When pump is active, suction is not_active.
2. When suction is not_active, pump is not_active.
3. When supply is active, pump is not_active.
4. When supply is not_active and suction is active, pump is active.
5. When pump is active, supply is active.

The programs utilise menu-driven user interfaces and supports problems of up to 16 states. Larger problems must be broken up into smaller ones. Weaknesses of this system are its lack of support for hierarchical decomposition of large problems and the limited functionality of the specification language.

Although many additional real-time programming languages have been developed that could be applied to process control, they are not commonly applied to PLC programming. These languages include: Ada, Industrial Real-Time FORTRAN, HAL/S, LTR, PEARL, Real-Time PL/1, and Real-Time Euclid (Halang, 1990). Since they are not used in programming PLCs they are not further considered in this investigation, nor were they considered important by the industry partner.

3.7 CASE Tool Architecture for PLC Software Development

Roberts presents a context for a methodology for the generation of relay ladder logic for control in discrete manufacturing from a high level specification based upon the Hatley and Ward/Mellor methodologies (Roberts, 1990). Figure 3.5 illustrates Roberts' proposed overall architecture.

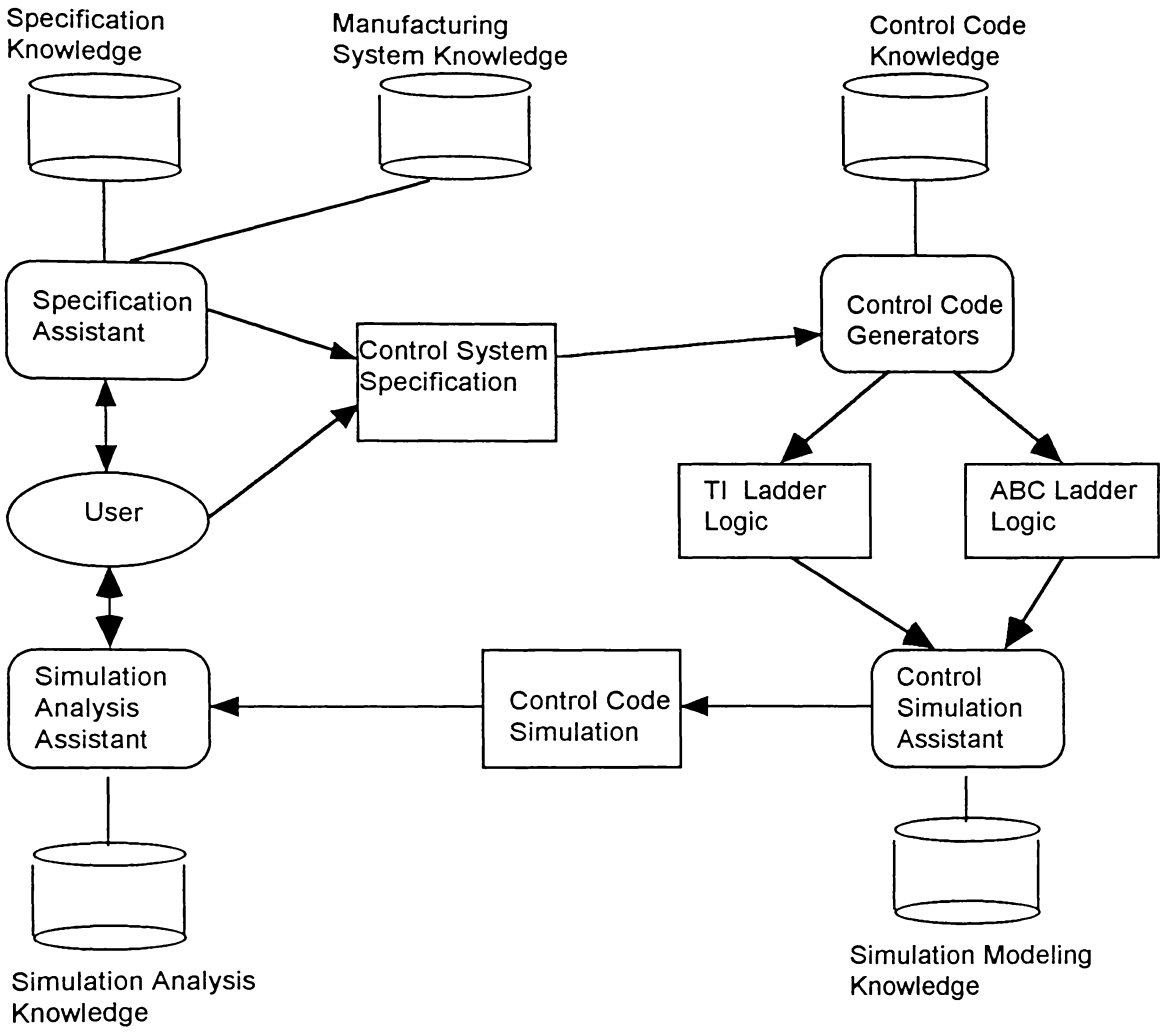


Figure 3.5. Proposed PLC Software Development Environment. (Roberts, 1990)

Using the above context as an overall approach, Roberts concentrates on the development of a graphic specification language and a mechanism for mapping the graphic specification into relay ladder logic for PLCs.

Roberts' environment incorporates many of the functional modules of the generic CASE tool architecture presented in Section 3.3. Furthermore, the use of specification and manufacturing knowledge in Roberts' environment bears resemblance to the concept of a methodology companion described by McClure (McClure, 1989; Vessey, 1992) and is thus a fruitful base on which to build CASE tools to support the development of PLC code.

As the generic CASE tool architecture and Robert's architecture demonstrate, an important component of a model CASE system is an automatic code generator. Automatic code generators for relay ladder logic were reviewed in Section 3.6. In the following section, the history and methods of automatic programming in general are reviewed.

3.8 Automatic Programming

Rich and Waters observe that although automatic programming has been a long term goal of computer science and artificial intelligence, the goal has been a moving target (Rich, 1988). In 1958, most "automatic programming" systems were in fact assemblers or FORTRAN compilers. Current research in the area includes: reusable software components, knowledge-based approaches, fourth generation languages and program transformation. The first three approaches are most applicable to this project, and are reviewed in this section.

3.8.1 Reusable Software Components

Reviews of the state of the art in research relating to *reusable software components* are found in (Stear, 1992; Krueger, 1992; Keebler, 1993). (Although software reuse is considered under automatic programming in this review, some authors consider automatic programming to be a category of reuse (Krueger, 1992; Keebler, 1993).) The primary focus of the reusable software component approach postulates a set of standard software modules that can be automatically combined into an application by a composition process. This approach requires an environment with mechanisms for (1) specifying a reusable object, (2) finding objects, (3) modifying objects, and (4) knowing how to use objects to compose new systems. Current research includes: (1) efforts that concentrate on designing software and systems so that software modules can be reused, (2) module cataloguing and repository systems, and (3) systems for examining existing modules to select candidates for reusability. The development of programming languages that support object-oriented approaches including the mechanisms of inheritance, polymorphism, and tools like class browsers are aimed at creating (and

finding, in the case of browsers) reusable objects. An example of current research is DARPA's Domain Specific Software Architecture Program which is research directed toward the creation of a workstation-based environment to support the development, maintenance, and upgrade of avionics systems through the reuse of large portions of well-designed and well-document software (Mettala, 1992).

Another approach to reusable components is presented by Neighbors and the Draco system (Neighbors, 1984). This approach aims to reuse analysis and design information in addition to programming language code. The Draco system suggests use of domain analysis, wherein a human, called a domain analyst, examines the requirements of a collection of systems that are similar, and describes the objects and operations that are relevant to that domain. The prototype Draco system developed by Neighbors successfully demonstrated the feasibility of the approach, but concluded that domain analysis, as idealised by Neighbors, is very difficult. For example, a domain analysis of tactical display systems is over 100 pages in length. Barstow notes that although the aim of the Draco system is an appropriate long-term research goal, the approach is not yet feasible because too little is known about how to organise and structure the domain knowledge and about how domain knowledge and programming knowledge interact (Barstow, 1985).

3.8.2 Knowledge-Based Approaches

A number of researchers are working to apply *knowledge-based* approaches to software generation -- this field has been called knowledge-based software engineering. McCartney makes the following observations concerning research in this area (McCartney, 1991): Real progress is possible in restricted domains, as opposed to general-purpose intelligent systems. Research in the field is on systems that perform tasks that are now performed by human programmers and designers, such as decomposing a task into sub-tasks and performing optimisations on code as opposed to a generalised intelligence system. Reviews of current research is presented in Lowry (Lowry, 1991; Stear, 1992).

Knowledge-based approaches have also been explored for use as expert assistants, as in the MIT Programmer's Apprentice project (Waters, 1985). Assistants are interactive programming environments that consist of collections of tools such as intelligent editors, on-line documentation aids, and program analysers (Rich, 1988). No major breakthrough in this area has been achieved yet (Stear, 1992).

Another application of knowledge-based techniques is to extend the capabilities of computer aided software engineering tools. The following synopsis is presented by McCartney (McCartney, 1991). Knowledge-base techniques can be applied to CASE in two ways: (1) by improving the knowledge base available in a CASE tool by using AI techniques for storing and inferring facts and (2) to fundamentally change the notion of CASE (McCartney, 1991). The former application is the simplest and is aimed at improving the performance and range of current CASE tools. The latter is a more fundamental change. It is aimed at: (1) Formalising the development process by providing a specification language and explicit set of refinements that can be applied at various stages of development; (2) Development and use of formal domain models to support the application area; (3) Changing the software process so that verification and maintenance are done on the specification rather than on the source code. This could include recording of design decisions and associated rationale and is consistent with the proposal by Baxter for the use of Design Maintenance Systems (Baxter, 1992).

The theorem-proving approach to knowledge-based programming is based upon the observation that a program could be derived from a formal specification and the use of techniques for automating proofs in first-order logic (McCartney, 1991). There are fundamental difficulties with this approach. The technique is only applicable to short programs, and the approach requires that the user provide a formal domain theory and specification, demanding skills not possessed by most programmers and end-users.

3.8.3 Fourth Generation Languages

Fourth generation languages are application-specific languages that capture the syntax and semantics for a particular problem area. The following analysis is presented

in (Stear, 1992). Fourth generation languages aim to: (1) Improve the support for the interface between the application domain and the user; (2) Allow the user to specify what needs to be done, and the system decides how; and (3) Minimise the number of visible steps in the software generation process. Although fourth generation languages are currently very successful in business applications, initial efforts are just beginning in the area of control systems.

3.8.4 Summary -- Automatic Programming

A common theme to all of the approaches described above is the use of *domain specific* knowledge and languages. The requirement that a successful automatic programming system be domain specific is also emphasised by Rich and Waters, and is summarised as follows (Rich, 1988). End-user-oriented automatic programming systems must be domain experts. Focusing on a narrow enough domain makes it feasible to construct a fully automatic program generator that communicates directly with end users. The crucial part played by domain knowledge is also echoed by Barstow who states that "Domain knowledge is crucial to an automatic programming system ... at the current time" (Barstow, 1985). Domain specificity permits both natural interaction with the end-user and supports the decision-making process (Barstow, 1991).

Another result is that researchers have found that there is no single best way to specify programs; neither graphics, examples, natural language, nor formal mathematical language representations worked best (Frenkel, 1985).

3.9 Automatic Programming for Programmable Logic Controllers

Automatic generation of PLC code has been demonstrated by several researchers. This work was reviewed in Section 3.6. To summarise, Ahmed has developed a limited-scope ladder logic generator for manufacturing systems that employ a number of pneumatic or hydraulic cylinders arranged to operate in a specific sequence (Ahmed, 1988). This system is narrow in scope, and does not address the domain under consideration in this project. Several researchers have developed general PLC code

generators that accept lists of operations and associated actions and conditions and generate the equivalent of ladder logic code. These include the work of Bhatnagar & Linn and Gunasena & Lehtihet. Both of the systems described above are based upon textual specification languages. They lack mechanisms for hierarchical decomposition, thus limiting them to small applications. Further, the specification languages are not executable or simulated.

Several projects are reported by Krogh and his associates at Carnegie Mellon University. These projects utilise the approach described above, namely to specify the control system by coding a list of operations, enabling conditions, and actions and then transforming these operations into either executable code or Petri nets which are then used to perform various forms of analysis (Pathak, 1989; Krogh, 1988; Krogh, 1987a; Krogh, 1987b; Ekberg, 1987; Beck, 1986).

An interesting automatic code generator for use with small problems is reported by Devanathan, which uses a restricted English textual language as input, generates a state transition diagram, which is subsequently translated to RLL code for input to a PLC (Devanathan, 1991). The input language is a sequence of 'cause' and 'effect' statements.

In summary, these projects have demonstrated the feasibility of producing some PLC control code from a higher level specification. Most of the specification approaches require that the designer specify the control system in terms of preconditions, actions, and in some cases, invariants. These specification languages are generic, not domain-specific, and lack facilities for hierarchical decomposition and ways to conveniently handle "global" conditions such as failure events. Moreover, most do not generate functional PLC code (with the notable exception of Devanathan's system), but may generate an equivalent representation, such as boolean logic or C.

3.10 Model-Based Analysis

Another important characteristic of modern CASE-based software development systems is support for analysis and simulation of a design specification prior to programming (or code generation). In fact, Baxter suggests that design maintenance

should become the key focus of the software maintenance process, with programs being derived from the complete design (Baxter, 1992). He proposed a tool called a design maintenance system (DMS) that provides the environment for this process. DMS requires that software be formally specified, its implementation be derived from the specification, the justification for the implementation be captured, desired maintenance changes be formally specified, and also necessitates suitable tools for capturing design justification and modification (Baxter, 1992). By deriving programs from the design, the software development life cycle will change to require that any code modifications at any point in the life cycle be fed back to the requirements specification stage, and then the program is re-derived from the specification (Sciences, 1989).

Other researchers who emphasise the importance of model-based analysis include Harel, who argues that a development environment for reactive systems requires a means for modelling the system and techniques for inspecting and analysing the model, and Kopetz who identifies support for evaluation of the design before initiating the test phase as one of the requirements for a process control development system (Harel, 1992; Kopetz, 1991).

In addition to simulation, another approach to program analysis is program visualisation. Program visualisation systems may be used to illustrate code, data, or algorithms and may illustrate dynamic or static characteristics of a program. Dynamic visualisations can show an animation of the program running, whereas static systems are limited to snapshots of the program (Myers, 1990).

The preceding sections concentrated on a review of various aspects of CASE technology and development methods for control software. The next section examines the areas of potential organisational impact of CASE tools.

3.11 Dimensions of CASE Technology

CASE tool technology can be conceptualised as aiding the life cycle through the following functionality (Henderson, 1990).

1. Production technology. This is functionality that directly impacts the capacity of an individual(s) to generate planning or design decisions and subsequent artefacts or products. Production technology includes support for product representation (eg. specification), analysis (eg., simulation, evaluation of alternative representations), and transformation (eg., code generation). These are defined as:

- Representation. Functionality to enable the user to define, describe or change a definition or description of an object, relationship, or process.
- Analysis. Functionality that enables the user to explore, simulate, or evaluate alternative representations or models of objects, relationships, or processes.
- Transformation. Functionality that executes a significant planning or design task, thereby replacing or substituting for a human designer/planner.

2. Coordination technology. This is functionality that enables or supports the interactions of multiple agents in the execution of a planning or design task. Coordination technology includes control (eg. resource management and access control) and cooperative functionality (eg., exchange of information). These are defined as:

- Control. Functionality that enables the user to plan for and enforce rules, policies or priorities that will govern or restrict the activities of team members during the planning or design process.
- Cooperative Functionality. Functionality that enables the user to exchange information with another individual(s) for the purpose of influencing (affecting) the concept, process or product of the planning/design team.

Cooperative functionality is identified as of prime importance by Winograd who suggests that individuals in an organisation, especially managers, are primarily concerned with generating and maintaining a network of conversations for action -- conversations in which requests and commitments lead to successful completion of work. Winograd suggests that a computer system's role is not one of a surrogate expert, but as an intermediary -- a sophisticated medium of communication which may in turn open new conversation networks that give the organisation the ability to recognise and realise new possibilities (Winograd, 1986).

3. Organisational technology. This is functionality and associated policy or procedures that determine the environment in which the production and

coordination technology will be applied to the planning and design process. Organisational technology includes aspects of support (eg., use of domain knowledge and on-line help) and infrastructure (eg., functionality standards that enable portability of skills and data). The definitions are:

- Support. Functionality to help an individual user understand and use a planning and design aid effectively. This includes functionality that uses domain knowledge.
- Infrastructure. Functionality standards that enable portability of skills, knowledge, procedures, or methods across planning or design processes. This includes adopting standard structures or languages for representing design information.

These three dimensions are summarised by the diagram in Figure 3.6. The position of organisational support below production and coordination technology indicates that it determines the environment of these other two areas.

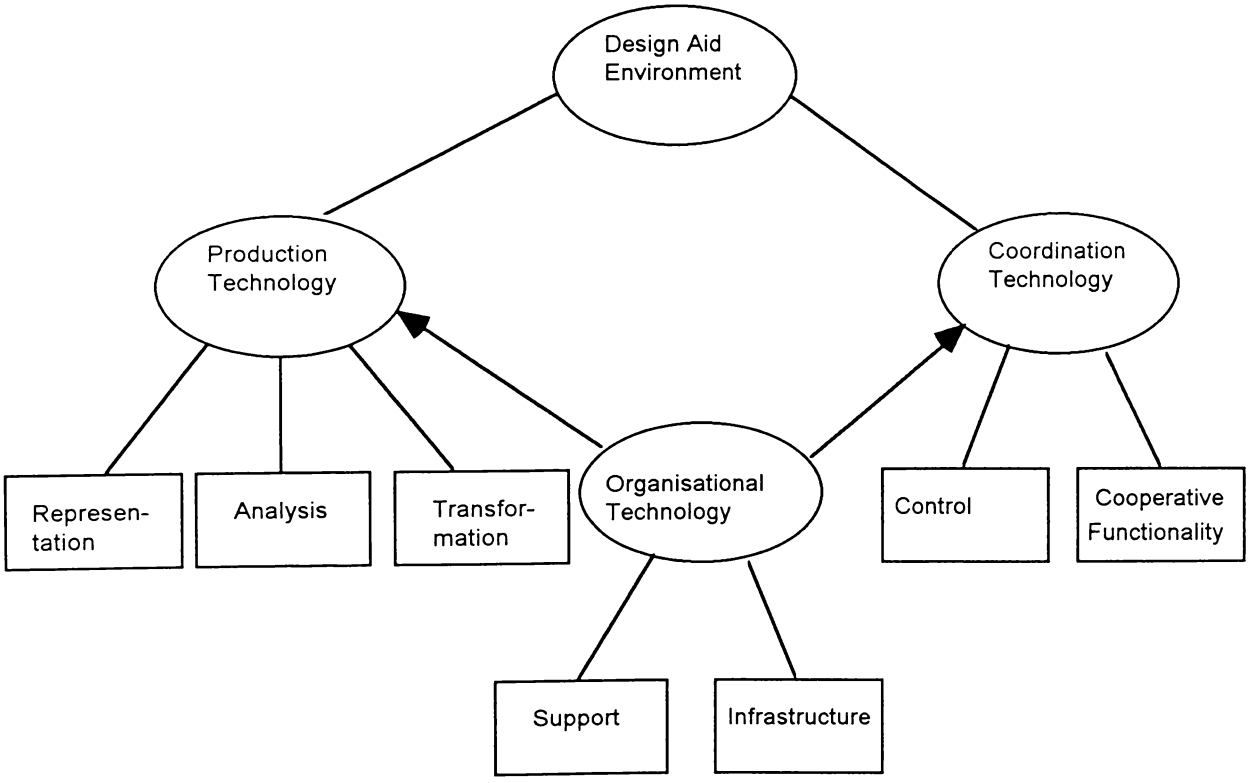


Figure 3.6. Dimensions of CASE Technology Functionality (Henderson, 1990).

In this investigation, these three areas of production, coordination, and organisational technology will serve as the basis for studying the potential influence of CASE technology on the development of manufacturing control software. In addition,

a global measure of effectiveness that will be used is that suggested by Winograd who notes that a system is effective if an organisation using it finds itself in a better position by using the system (Winograd, 1986).

3.12 Summary

CASE tools, environments, and the categories of toolkits, workbenches, and methodology companions were introduced. The promise of CASE and theories for the inconclusive evidence of the acceptance and impact of CASE tools were also reviewed.

A definition of domain specificity as it applies to CASE tools was developed that examines tools in terms of abstraction and specialisation. Abstraction has two levels, the abstraction specification and the abstraction realisation. The abstraction specification is further broken down into variable, fixed, and hidden parts. The larger the fixed and hidden parts, compared to the variable part, the easier a tool is to use because it requires less specialisation. An intuitive measure of the effort to specialise a tool is called cognitive distance. A domain-specific tool is one with a small cognitive distance that requires less specialisation than generic tools that require extensive specialisation (they have large cognitive distance). Thus domain-specific tools seek to maximise the fixed and hidden parts of the specification while minimising the variable parts through built-in methodology support and automation.

A set of steps to guide the development of application generators, which are highly domain-specific, was identified. These steps call for definition of the domain and its boundaries, identification of an underlying model (abstraction), definition of the variable and fixed parts of the abstraction, identification of the user interface, definition of the products of the generator, and implementation of the generator. In Chapter 5, these steps will be combined with the process of domain analysis and the definition of domain-specificity to yield a methodology for guiding the development of domain-specific CASE tools.

Generic real-time methods for specification of real-time control systems were reviewed and it was found that most employ the three modelling abstractions of

functional, structural, and behavioural views. The most common specification methods include data-flow techniques for functional models, graphical templates and entity-relationship diagrams for structural models, and finite state machines for behavioural models. Less common models include Petri nets or modifications of Petri nets. Examples of methodologies and tools that employ these methods were also reviewed.

Specification methods and tools targeted specifically toward development of PLC code were presented. The most common specification method is to build a data base of pre-conditions, actions, and post-conditions. These are then translated into intermediate models such as Petri nets or into relay ladder logic, boolean form, state diagrams, C, or Ada. A graphic notation was proposed by Roberts based on the Hatley and Ward-Mellor methods, and he also described a conceptual architecture for PLC development and automatic code generation.

Proposed language standards for programming PLCs were reviewed. The most common language is relay ladder logic.

Features that should be incorporated into CASE tools were identified in the literature. These are support for model-based analysis, design maintenance, and automatic code generation. Researchers call for the modelling and maintenance of the design to become the focus of software development, with the executable code automatically generated from the specification.

Finally, the potential areas in which CASE tools may impact an organisation were identified and will be used in this investigation to study the influence of CASE on the development of control software.

The next chapter reviews the domain of control software for batch manufacturing. To define the context of the investigation, the industry partner is introduced and an analysis of current practice is described.

Chapter 4 Problem Domain

This chapter is a domain analysis of the field of control code development for batch manufacturing, both in general and at the industry partner in particular. The software development life cycle and current practice at the outset of this study are reviewed, with a focus on problems that may be addressed by CASE technology.

4.1 Industry Partner

This investigation is set in the context of an operational batch process manufacturing environment at Anchor Products Ltd., a member of the New Zealand Dairy Group of companies. In the 1992-93 financial year, the New Zealand Dairy Group reported turnover of $2,007.2 million. Up to 21 million litres of milk per day are processed through the company's ten factories, which accounts for nearly half the total volume of milk produced in New Zealand. Over ninety percent of the products it makes are exported around the world, and those exports account for over ten percent of New Zealand's total export revenue (Anchor, 1993). In addition, Anchor has made major additions to plant capacity in each of the last two years.

Anchor makes extensive use of automated batch process control, using programmable logic controllers, in the production of dairy products at its large processing plants. The author (researcher) and the engineering management at the Anchor agreed to enter into a collaborative relationship to support the author's research goals and attempt to improve the practice of control software development in that industry.

Prior to the design of the prototype CASE environment, a domain analysis was performed to document the existing practice, terminology, and procedures utilised in batch process manufacturing in general and at Anchor Products in particular. As noted in the literature review, information about the domain can be found in the literature as well as within the particular organisation. The results of this study are described below.

4.2 Definition of Batch Manufacturing

The following description of batch process manufacturing is from (Shaw, 1982). Shaw classifies a manufacturing process as batch if, due to physical structuring of the process equipment or due to other factors, the process consists of a sequence of one or more "steps" (or "phases"), in which ingredients are mixed, cooked, reacted, finished, or otherwise processed, and must be performed in a defined order. The completion of the sequence of steps creates a finite quantity of finished product. The sequence of steps must then be repeated to create additional quantity of the product. Figure 4.1 illustrates the batch model of manufacturing.

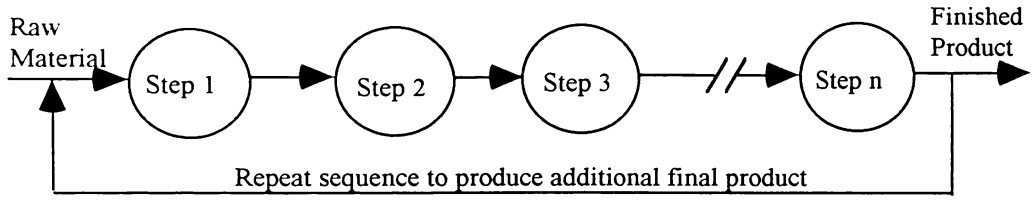


Figure 4.1. Model of Batch Manufacturing (Shaw, 1982).

The steps within a batch process consist of one or more operations performed on the ingredients. Generally a step, once started, must be completed in order for processing to be done without damage to the raw product; only completion, error conditions, called faults, or operator intervention cause an exit from a step.

The flow of control from one step to the next is generally "gated" by conditions to permit operator intervention and automatic detection of faults. Figure 4.2 shows the typical logic for a step.

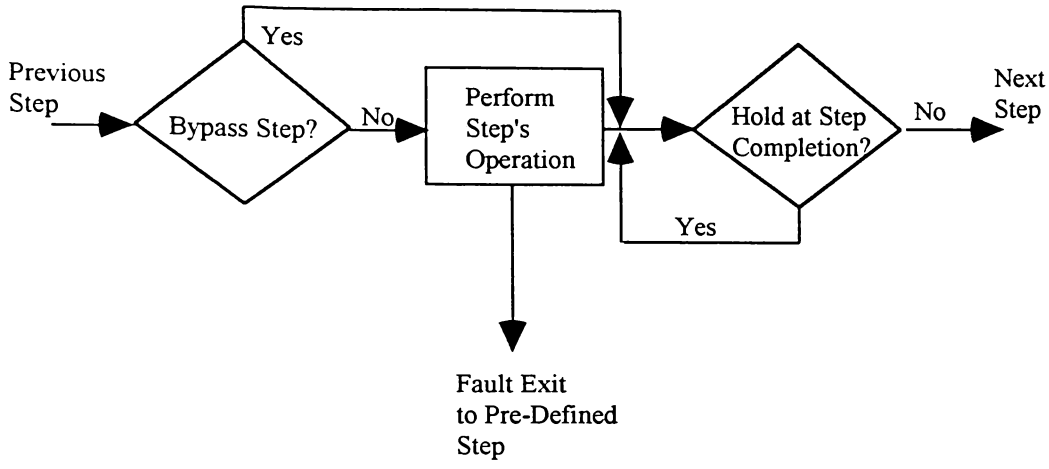


Figure 4.2. Typical Process Step (Shaw, 1982).

The bypass conditions allow steps to be terminated and the normal sequence of steps to be altered. The hold points between steps allow the operator to control continuation of the process. The automatic fault logic allows for special conditions arising from faults to be detected and processed, typically by stopping the normal process and initiating alarms.

In automated batch manufacturing the processing steps are controlled by a computer such as a PLC.

4.3 A Representative Structural Model for Batch Processing Control

Although there is currently no universally accepted standard for describing batch systems, some efforts have been made to produce standard models and terminology. The International Purdue Workshop on Industrial Computer Systems, Special Applications Programming Committee TC-4 worked for several years to develop a structural model and terminology for batch control systems (Habib, 1990; Barb, 1987; Bristol, 1985). Similar work has been done by the NAMUR Committee in Europe (Fisher, 1990). Both of these groups are now inactive, but in October 1988 the Instrument Society of America (ISA) established a committee, SP88 Batch Control Systems, to provide standards and recommended practices for the design and specification of batch control systems (Bunch, 1994). It's output to date is based

heavily on the work of the Purdue TC-4 and NAMUR efforts (Fisher, 1990). The following model of batch control, taken from (Fisher, 1990) is based upon this work.

The processing actions of a batch system tend to be organised into a hierarchy of five levels as shown in Figure 4.3.

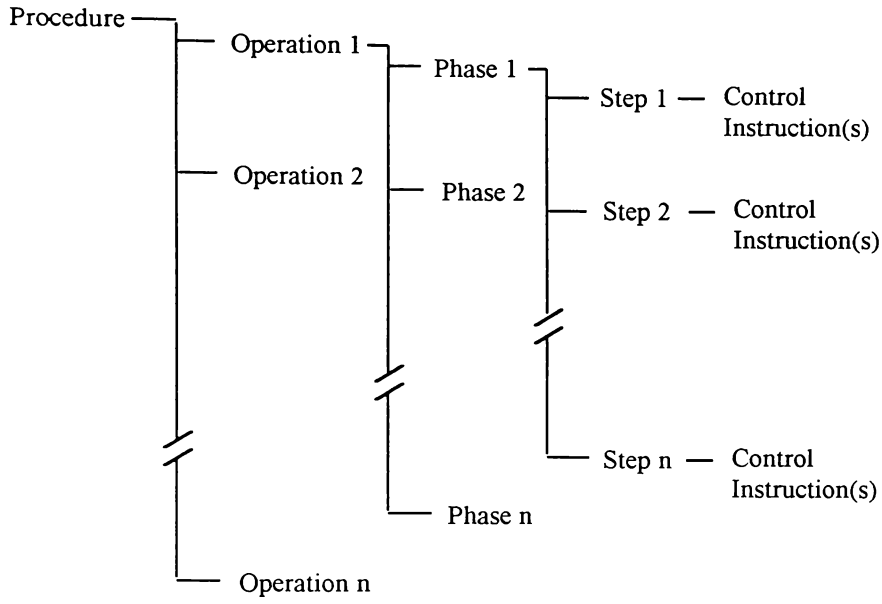


Figure 4.3. Process Model for Batch Processing (Fisher, 1990).

A procedure defines the generic strategy for producing a batch of a product. Procedures define and order the actions to be performed and the control requirements for making a general class of products.

Procedures consist of operations that are carried out on a collection of associated equipment called a *unit*. A unit is the primary object for automatic control of a batch system; it is connected to the control system through its valves and other actuators, and is monitored by the control system through various sensors. Some examples of operations on units are changing, reacting, distilling, and neutralising.

Operations consist of phases. A phase is defined by boundaries that constitute safe or logical points where processing can be interrupted to, for example, allow for operator intervention to change an ingredient or the flow of control of the process. The sequence under which the phases are controlled constitute the batch operation. Figure

4.4 illustrates the relationship between operations, phases, and steps (steps are described below).

Operations	Phases	Steps
Charge	Add Ingredient A	
		Open Valve A
		Check Weight in Tank A
		Close Valve A
	Add Ingredient B	
	Mix	
	Add Ingredient C	
React	Heat	
	Add Ingredient D	
	Hold	
Pumpout	Cool	
	Transfer	

Figure 4.4. Example of Operations, Phases, and Steps (Fisher, 1990).

The control steps are the lowest level specification that describes an event or action that is of interest to an operator.

Steps typically involve one or more commands to the control elements of the factory. (An element is a piece of plant equipment having an active function such as measuring or manipulating a physical attribute.) In terms of a high-level programming language, a control step would map to a statement in the programming language. For example, the control steps involved in the phase "Add Ingredient A" are shown in Figure 4.4 to be Open Valve A, Check Weight in Tank A, and Close Valve A.

Comparing Shaw's model described in Section 4.1 to Fisher's model in this section, Shaw's steps correspond to Fisher's phases and thus Shaw's model is simply one level of Fisher's hierarchy.

The next section describes a model language for specification of the step logic.

4.4 A Representative Language Specification of Batch Control Steps

Shaw presents a language for specification of the logic of batch programs (Shaw, 1982). A synopsis of this language is reviewed in this section.

The language is composed of commands that fall into six categories:

- commands that manipulate devices
- commands that manipulate continuous controls
- commands that manipulate timers
- commands that permit subroutines
- general purpose commands
- program control commands

Examples from each category are described below.

Three types of *device commands* are specified. These are:

- Device Driving Commands. Commands to set a device to a specified state.
- Device Override Commands. Commands to disable fault checking of the device input state to its output state, which is normally performed.
- Device Check Commands. Commands used to test a device state.

Examples of these three kinds of commands are shown in Table 4.1.

Command	Type	Description	Example
OPEN/START/ON	Driving	Open, turn on, or start a device	OPEN VALVE2
CLOSE/OFF/STOP	Driving	Close, turn off, or stop a device	CLOSE GV12
MID/HALF/IDLE	Driving	Drive a 3-state device to an intermediate state	MID FV200
ONOVRD	Override	Set device override to ignore fault checking	ONOVRD PUMP
OFFOVRD	Override	Clears override	OFFOVRD AGIT1
CHKOPEN/ CHKON / CHKSTART	Check	Check that a device is OPEN/ON/ STARTED	CHKON PUMP2
CHKCLOSE/ CHKOFF/ CHKSTOP	Check	Check that a device is CLOSED/OFF/ STOPED	CHKSTOP RV15

Table 4.1. Examples of Device Driving Commands.

Continuous control commands monitor and manipulate values, state, and alarm conditions associated with continuous control devices. Continuous control devices regulate flow, temperature, or pressure in a vessel on the basis of a *ramped set point* using a *loop*. A loop is a combination of two or more pieces of plant equipment arranged so that signals pass from one to another for the purpose of regulating a process variable (Fisher, 1990). Continuous control commands can be divided into the following groups.

- Loop Put Commands. Commands to store the values and set the status of loop control variables. Values may be expressed as either numeric values, for example 0 - 8192 where 0 represents 0% of the loop's engineering units span and 8192 represents 100%, or simply as percentages.
- Loop Get Commands. Commands to retrieve the values and status of a loop.

- Loop Alarm and Control Status Check Commands. Commands to test for loop alarms or status values. If the specified condition is true, the program could branch to another statement.

Examples of continuous control commands are shown in Table 4.2.

Command	Type	Description	Example
PUTSETPT	Loop Set	Set a loop's setpoint value	PUTSETPT 1600, TIC143B
PUTAUTO	Loop Set	Change a loop's control status to automatic	PUTAUTO F200
PUTMAN	Loop Set	Change a loop's control status to manual	PUTMAN F200
GETMEAS	Loop Get	Get the process measurement value from a loop	GETMEAS TC250
IFDEVN	Loop Alarm	Check loop for setpoint vs. measurement deviation	IFDEVN GV42,ALARM2
CHKAUTO	Loop Status	Check a loop for automatic control status	CHKAUTO PIC350,ALARM3

Table 4.2. Examples of Continuous Control Commands.

Timer commands reset and start or stop timers. Timers are identified by a name or number and can count up or down in one-second increments. An example of a timer command is "Start timer TM50" which would set the timer TM50 to zero and start it incrementing. The command "TMRSTOP TM50" would stop, but not change, the value of the timer.

Subroutine commands can be used to branch to other program statements. An example, taken from BASIC, would be use the of GOSUB and RET to invoke and return from a subroutine.

General purpose commands provide for conditional execution, repetition, logical operations, and general mathematical expressions. Examples could include IF statements for conditional execution, WHILE and FOR loops for repetition, assignment statements, and generalised relational and mathematical expressions.

Program control commands allow execution of other programs (operations or procedures). Examples could include RUN, ABORT, PAUSE, and HOLD.

This section has presented a representative batch control language. The next section describes the batch language in use at Anchor Products.

4.5 Specification Language in Use at Anchor Products

At Anchor Products, a document called the *functional description* (FD) embodies the process engineer's specification of the automated plant's controls. It has evolved over the past twenty years in the dairy processing industry in New Zealand and at Anchor Products. The FD is used primarily by the PLC software engineer as the specification for development of the PLC software. Later in the life cycle it is used for testing, commissioning, and by plant floor personnel for maintenance and troubleshooting.

The FD is composed of several sections, which will be listed shortly. Of primary importance to this investigation is the portion of the FD, called selections, that specify the batch control logic. Anchor's method for specification of the control logic, illustrated in Figure 4.5, is quite similar to the structural model described in the literature and presented in Section 4.3.

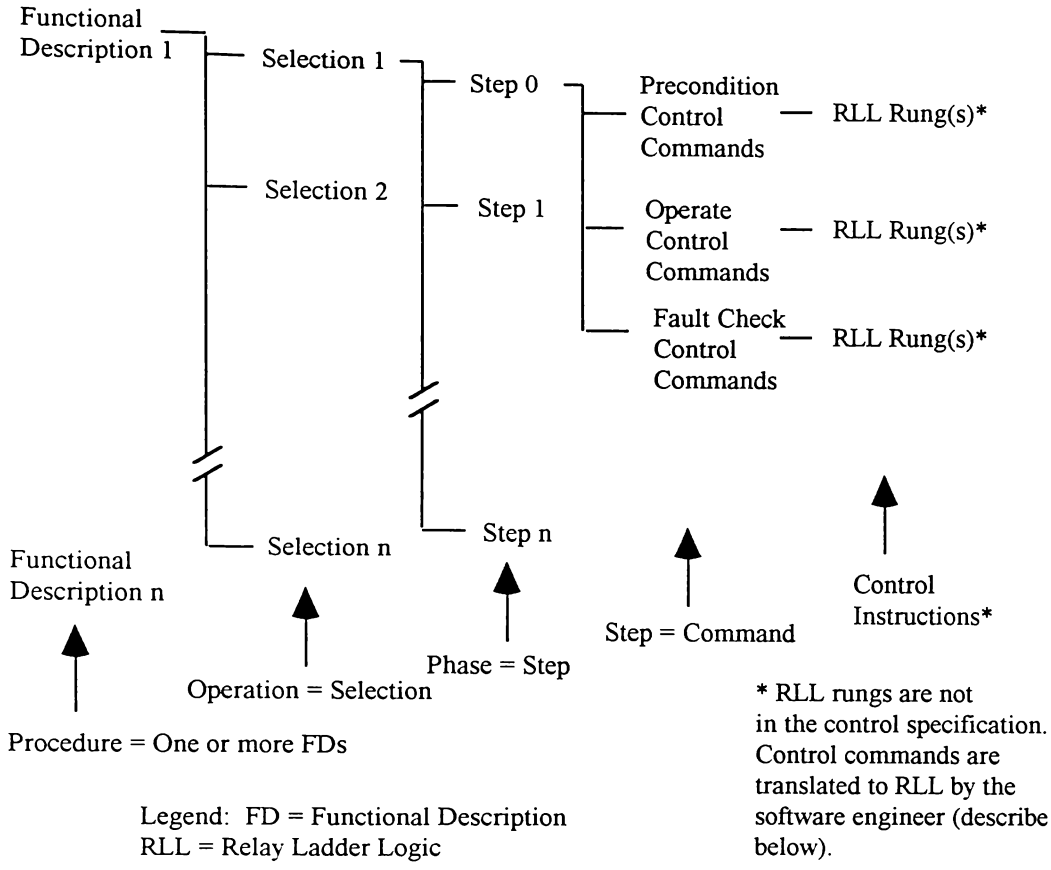


Figure 4.5. Anchor's Control Specification Structure vs. Model Structure.

A comparison between the model control specification structure (Figure 4.3) and Anchor's structure (Figure 4.5) reveals the mapping illustrated by the arrows in the latter figure. Notice that a "procedure" in the model maps to one or more of Anchor's FDs. Notice also that a step in the model maps to a control command (similar to those described in Section 4.4) and that, at Anchor, control commands are later translated into RLL code for execution by the PLC. (The RLL code was illustrated in Figure 4.5 to complete the hierarchy; RLL is not a part of the FD document.)

To support this research, Anchor provided the researcher with a model "Sample Functional Description" which was to be representative of a typical FD (see Appendix A), as well as examples of actual FDs. The model FD consisted of the following sections:

1. A copy of the plant and instrumentation diagram (P&ID) and a definition of the symbology.
2. A high level description of the process.
3. A listing of the automation database. This is a list of plant items, their types, and tags.
4. Selections, which comprise the control logic. Each selection is a sub-process (or operation) of the overall manufacturing process.
5. A listing of software tags, timers, and analogue switch points.
6. Reporting requirements.
7. Operator Interface requirements.

The specification language utilised in the sample FD for the selections, steps, and control commands will now be examined.

A *selection* is an independent production activity and corresponds to an operation in the representative structural model. In the sample FD, a selection named "Milk Transfer Silo 1 To Silo 3" is shown. In addition to being identified by name, selections in a FD are numbered, for example P101, P102, etc., sequentially, but this numbering does not imply order of execution. More than one selection may be active at a time.

Each selection is made up of one or more sequentially numbered *steps*, starting with step zero. (Recall that at Anchor steps correspond to phases in the representative structural model.) Steps are structured quite similarly to the typical process step (Figure 4.1) described by Shaw with one important difference: their flow of execution is not strictly sequential. Steps typically start in a sequential order, but more than one step within a selection may be (and usually is) active concurrently. Figure 4.6 shows the order in which the steps from a typical selection (an abbreviated example based upon Appendix A) will start their execution.

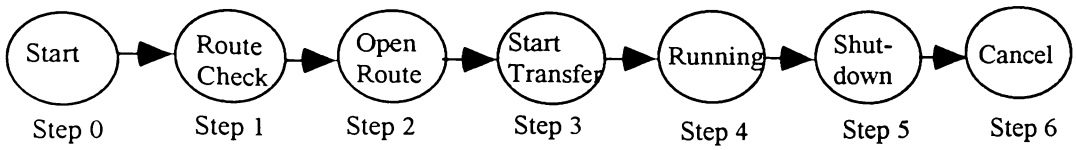


Figure 4.6. Order of Step Initiation (from example in Appendix A).

Figure 4.7 is a timing diagram that illustrates the concurrency in the step execution. In this diagram the term "active" is used when the step is running and "not active" indicates that it is not running. The arrows indicate events that activate or deactivate the steps.

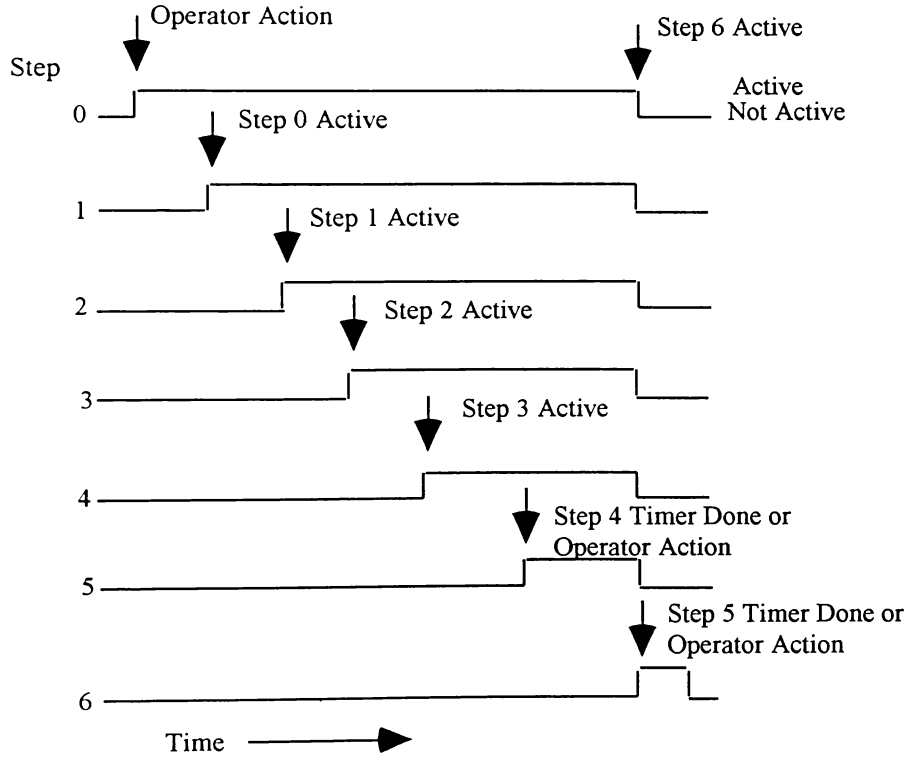


Figure 4.7. Concurrent Execution of Steps.

Figure 4.7 illustrates that, for example, step 0 continues to be active during the execution of steps 1 through 5. At one point, steps 0 through 5 are active concurrently and all terminate with the activation of step 6.

At Anchor, steps are composed of control commands, which are grouped into preconditions, operates, and checks. Preconditions and checks consist of device check, loop status commands, and timer checking commands. The operates contain the device

driving, loop put, timer starting and stopping, and program control commands. Figure 4.8 illustrates the structure of a step.

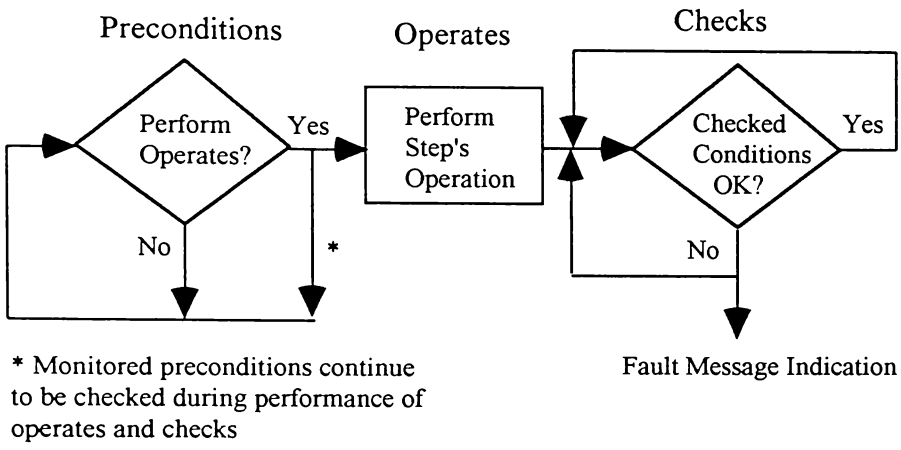


Figure 4.8. Structure of a Step.

All steps have *preconditions*, and may have *operates* and *checks*. The flow of control is dictated by the preconditions associated with each step. Preconditions are a set of predicates that must be true for the step's operates to be activated. These conditions may be events (one-shot conditions) or may be monitored (continuous actions). Operates are actions that are performed as long as all preconditions are true. Checks are conditions that are evaluated after the operates have been applied. If any check is false, this is called a fault and results in a message to the operator. The operates of a step with a fault continue to be active (as long as the preconditions are all true).

The command language for expressing the conditions for preconditions, the operate actions, and the fault checks is very similar to the representative language specification of Section 4.4. An example of a simple step and the command language is illustrated in Figure 4.9.

Step 3 (Start Transfer)

If the transfer route is confirmed open, the transfer pump is able to be operated, starting the transfer.

Step 3 Preconditions

1. Step 2 Active .And.	monitor; Route opened
No Faults Step 2 Checks	monitor; Route confirmed open

Step 3 Operates

1. Start P201	transfer pump
2. Enable FIC504 on Auto at FIC504_SP1	flow controller automatic; set pt FIC504_SP1

Step 3 Checks

1. P201 Running	transfer pump

Figure 4.9. Simple Step and Control Commands.

The flow of control in this step is shown graphically in Figure 4.10.

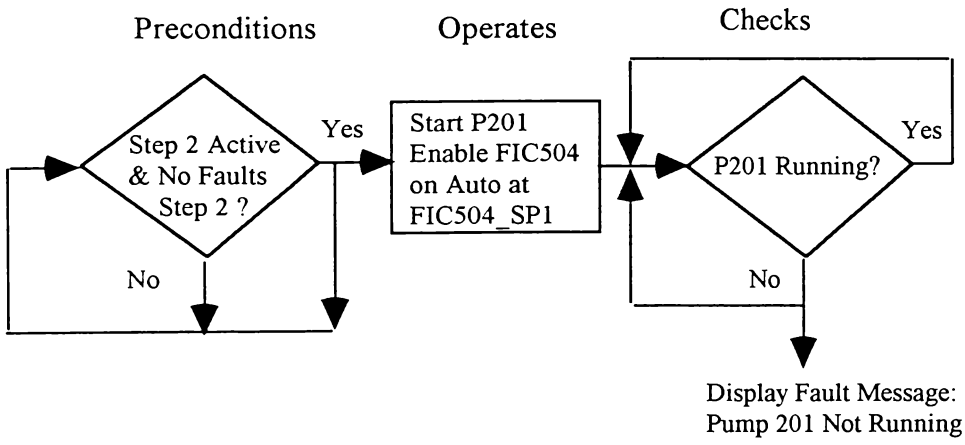


Figure 4.10. Flow of Control in Simple Step Example.

When a selection is started (also called "selected"), all of the steps in the selection will initiate their precondition checks. If the preconditions of any step are satisfied, it's operate commands will execute. Following execution of the operate commands, the conditions in the checks are tested. If any of these conditions are false, this indicates a fault in the operation of the step, and a message number is typically stored in the PLC's memory for later display (in the form of a message) to the operator. Thus, a step can be in one of four states: Inactive, precondition, active, or fault state. A step is inactive if

the selection of which it is a part has not been started. If a selection has been started, the step will be either in the precondition, active, or fault state. The precondition state means that one or more of the conditions have not been satisfied and the operates have not been applied. If all preconditions are true and the operates applied and no subsequent fault detected, then the step is considered to be active. The fault state indicates that one or more fault conditions have not been satisfied. If subsequent to activation of the operates any of the monitored preconditions becomes false, then the operates will no longer be applied and the step will return to the precondition state. In the fault state, the operates continue to be applied and a message number indicating a fault message is stored in the PLCs memory for access by the man-machine interface (MMI). (As was mentioned in Chapter 1, PLCs typically operate without operator intervention. However, a PLC may communicate with another system, called the MMI, which permits operators to monitor the status of the controller and also to send commands to it for manual intervention.) In the example step shown in Figures 4.9, the operates will be enabled and the checks will be examined as long as the preconditions are true (since they are designated as *monitored* preconditions). If the check (P201 running) is not true, a fault in the step is indicated.

The example in Figure 4.9 also illustrates the command language. *Start* is an example of a device driving command and *Enable* illustrates a *loop put* command. The command *Running* is a device check command, and the commands *Step 2 Active* and *No Faults Step 2 Checks* can be considered to be check commands as well.

Other similarities between Anchor's command language and the representative language are the use of conditional commands, timer commands, and program control commands. Figure 4.11 shows conditional and timer commands. In this example, the devices LT501 and LT502 are level indicators, FT503 is a flow control totaliser, and Timer 101/3 is a timer name. LT501_L4, LT503_F1, and LT502_L1 are symbolic names for level or flow quantities.

```
If (Level LT501 < LT501_L4  .And.
    Flow FT503 < LT503_F1     ) .Or.
    Level LT502 > LT502_L1
then:
  Start Timer 101/3
```

Figure 4.11. Example of Conditional and Timer Commands from the Sample FD.

Program control commands (from other Anchor FDs studied by the author) are illustrated in the example in Figure 4.12.

```
If P101 Not Selected
then:
    P101 Preselect
    P101-103 Preselect
    Status display message        "Pre-selected"
```

Figure 4.12. Program Control Commands from the Sample FD.

In this example, the "if" conditional examines the state of selection P101. If P101 is not selected (active), then process P101 is placed in the "preselected" state, the common selection group P101-P103 placed in the "preselected" state, and a status message ("Pre-selected") is to be displayed to the operator. In the Anchor methodology, selections can attain the following states: not active, preselected, or selected. When a step is not active, all its steps are not active, and none of the operates can be enabled. When a step is preselected, it is ready to run, but must wait until some other selection in its "common preselect" group finishes. The preselection state allows for automatic starting of selections from other selections. The selected state means that the selection is running, that is, its steps are either in the precondition, active, or fault state. Another example of a program control command is the command "Cancel P101" which would terminate all active steps in selection P101 and make that selection's state not active.

The structure and language used by Anchor for specification of batch control resembles closely the models described in the literature. Particular aspects of the Anchor's methodology will be examined further in Chapter 6 where the prototype CASE tool software is described.

The next section examines relay ladder logic coding conventions in use at Anchor.

4.6 Relay Ladder Logic Coding Conventions

This section presents an analysis of the relay ladder logic conventions employed at Anchor.

The RLL code is divided into different sections, called files. These files include:

- Files for each selection, composed primarily of code for preconditions and fault checks.
- Files for each type of discrete output device (such as VP, P, etc.) and loop (such as FIC) which contain the RLL code to implement operates for each of these devices.

The code in these files is described below.

The code for the selections is made up of the logic for each step. As an example, the RLL code that corresponds to the step from Figure 4.9 is shown in Figure 4.13.

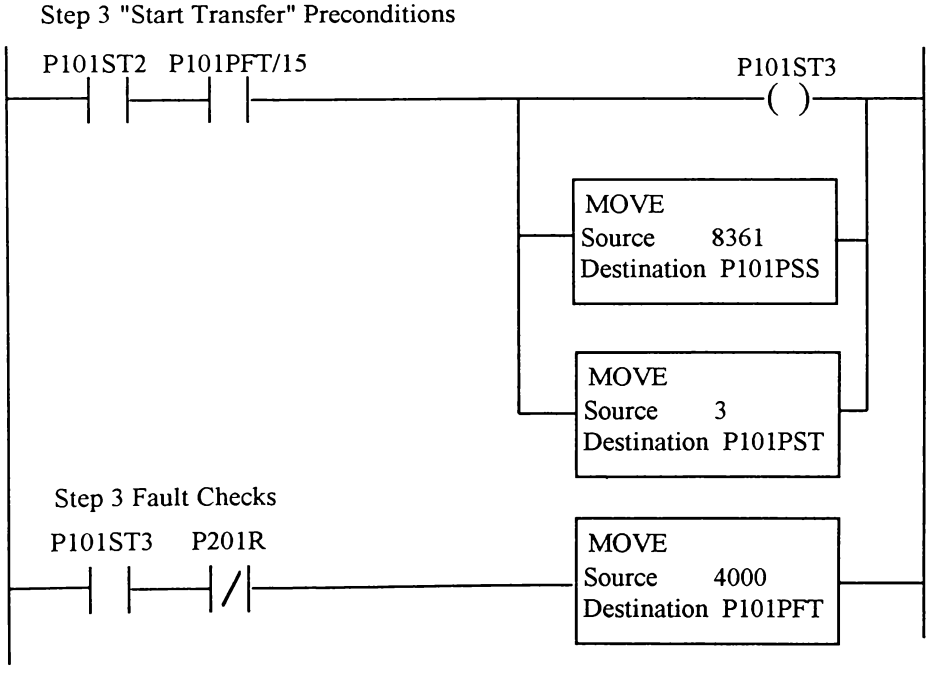


Figure 4.13. RLL Code Corresponding to Figure 4.11.

The preconditions (Step 2 Active .And. No Faults Step 2 Checks) are tested in the top rung and result in setting of a software flag that indicates whether or not the preconditions are true. This convention is used for each step, with the flag labelled

according to the selection and step numbers -- in this example, the flag is P101ST3 (Process 101, Step 3). In addition to setting the step flag, the conventions also call for storage of a status message number (in this case 8361) into the variable P101PSS (Process 101 Process Status) and the storage of the highest active step number (in this case 3) into the variable P101PST (Process 101 Process Step). The message number 8361 is mapped to the text string "Start Transfer" by the man-machine interface which is outside of the PLC. Notice the preconditions (top rung) for the step in Figure 4.13: the flag P101ST2 is tested and the high order bit (bit 15) of the P101PFT is checked to see if it is on. The first condition indicates that Step 2's preconditions were satisfied and the second condition indicates that there is no fault message currently stored in the fault message number variable. (Not shown is the convention that the variable P101PFT is initialised to -1 at the start of the selection, to indicate no faults.) If a fault is later detected, then a positive message number is stored in this field, thus clearing the sign bit of the fault message value as is illustrated in the second rung, where a fault message number (8361) is stored.

The second rung in Figure 4.13 illustrates the RLL code for the fault checks. The first condition is to determine if the step's preconditions were satisfied (by testing flag P101ST3). The second condition is the actual check that examines the state of pump P201's running feedback signal, labelled P201R. If this signal is not enabled, then a fault is indicated by storing the message number 4000 (which is mapped to the text string "Pump 201 Not Running") into the fault message variable P101PFT.

Notice in Figure 4.13 the absence of RLL code for the operate part of the step. In most cases, the code for the operates, such as device driving commands and analogue loop commands, are stored in separate RLL files. Anchor uses the convention of gathering all driving commands for similar types of plant devices into separate files. Thus, for example, all operates to drive valves of type VP are collected into one file, all operates for pumps into another file, all operates for analogue loops into another, etc. Figure 4.14 illustrates the RLL code for the driving command (operates) for pump P101.

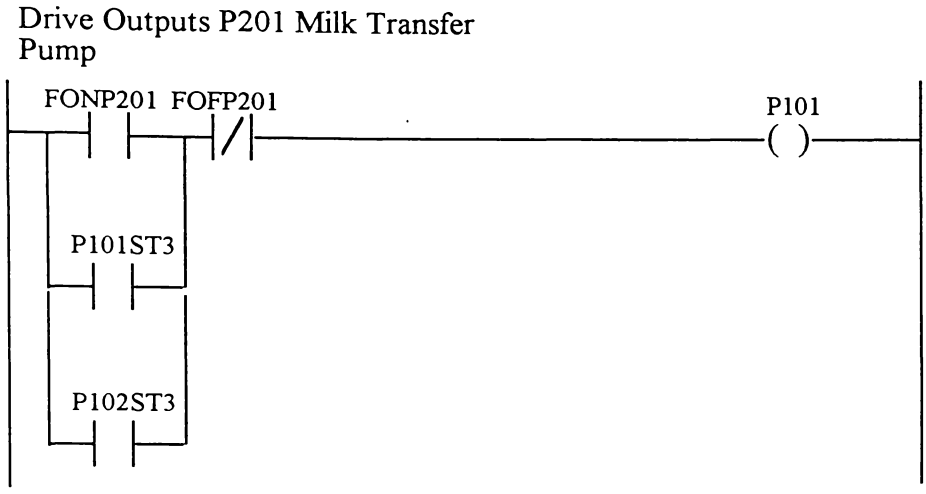


Figure 4.14. RLL Code for Operates to Drive the Pump.

This code says that pump P101 may be actuated if either step 3 from selection P101 (flag P101ST3) is active or step 3 from selection P102 (flag P102ST3) is active. (Thus in this example, the pump is referenced by operates in two selections: P101 and P102.) Figure 4.14 illustrates two additional flags that are part of the Anchor's coding convention: FONP201 (Force On P201) and FOFP201 (Force Off P201). These "force" flags allow for operator intervention to override the automation system to force the pump either on or off at operator discretion. The logic for this code is:

P201 = (FONP201 OR P201ST3 OR P102ST3) AND NOT FOFP201

When bit P201 is on, the pump will be started. Thus, this logic says that if either the force on bit is enabled, or Step 3 is active in selection P101, or Step 3 is active in P102 and the force off bit is not enabled, then run the pump. Using this coding convention, the software engineer can find, in one RLL file, all conditions that enable pump P201.

The examples in this section reveal that there is RLL code included in the PLC that is not explicitly specified in the FD. For example, the control logic does not call for storage of status and fault messages, storage of the current step number, nor the use of the force on and force off bits. The code for these requirements are added by the software engineer based upon experience or may be documented as narrative text in the reporting requirements section of the FD.

4.7 Software Development Life Cycle

The development of control software in this industry operates under severe time constraints and pressure. Final financial decisions to proceed with major plant upgrades, requiring development of control code, are often made only a matter of four to six months before the work must be complete and operational. Typical cycles might allow two to three months for development of the software specification, and two months for RLL code development and bench simulation testing. Plant upgrades are done in an intensive period from late June to late July during the period when New Zealand milk production is minimal. This leaves about a two week window for in-plant installation, final testing, and commissioning before the plant must be operational.

Other constraints on the development process include the requirement to use a particular brand of PLC hardware for maintenance or other reasons, and the requirement of certain company standard approaches as to how the PLC software is structured and coded.

A high-level view of Anchor's idealised PLC software development life cycle is illustrated in Figure 4.15.

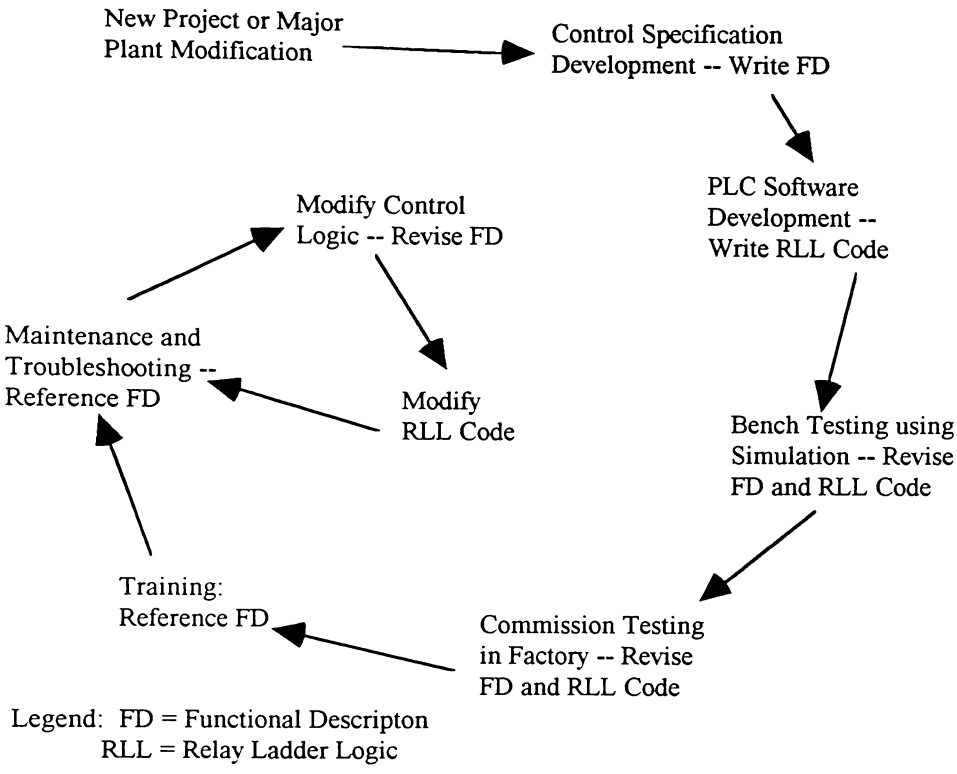


Figure 4.15. Idealised PLC Software Development Life Cycle.

At Anchor new PLC software development projects usually start with the decision to build a new factory or to do a major upgrade of an existing factory. Tenders would be sought from primary process contractors, who in turn would sub-contract the purchase of PLC hardware and development of PLC software to process engineers and software developers. For a given project, a number of parties might be involved in some aspect of the specification or creation of the software:

- Original Equipment Manufacturer (OEM) project manager and engineers. Contracted by Anchor to install process manufacturing hardware and control software.
- Anchor management. Selects the OEM contractor and sets engineering standards.
- Process automation engineering consultant. Subcontracted by the OEM contractor to write the process control FD which serves as the PLC code specification.
- PLC software engineer consultant. Subcontracted by the OEM contractor to translate the FD process control specification into the PLC RLL code.

- PLC supplier. Subcontracted by OEM contractor to deliver PLC hardware and possibly software design and development assistance.
- Electrical contractor. Subcontracted by the OEM contractor to install wiring (I/O points) between factory devices and the PLC. Supplies the I/O addresses needed by the software engineer.
- Anchor maintenance electricians. After commissioning of a new system, these electricians maintain the automation controls, including the software for minor modifications.
- Anchor software engineers. After commissioning of a new system, these engineers may develop or modify the PLC RLL code for more extensive factory modifications.
- Anchor process operators. Operators run the day-to-day operation of the factory. They may refer to the FD for troubleshooting problems in the factory.

Incorporating these parties into the life cycle diagram yields the Figure 4.16, to show how these parties are involved.

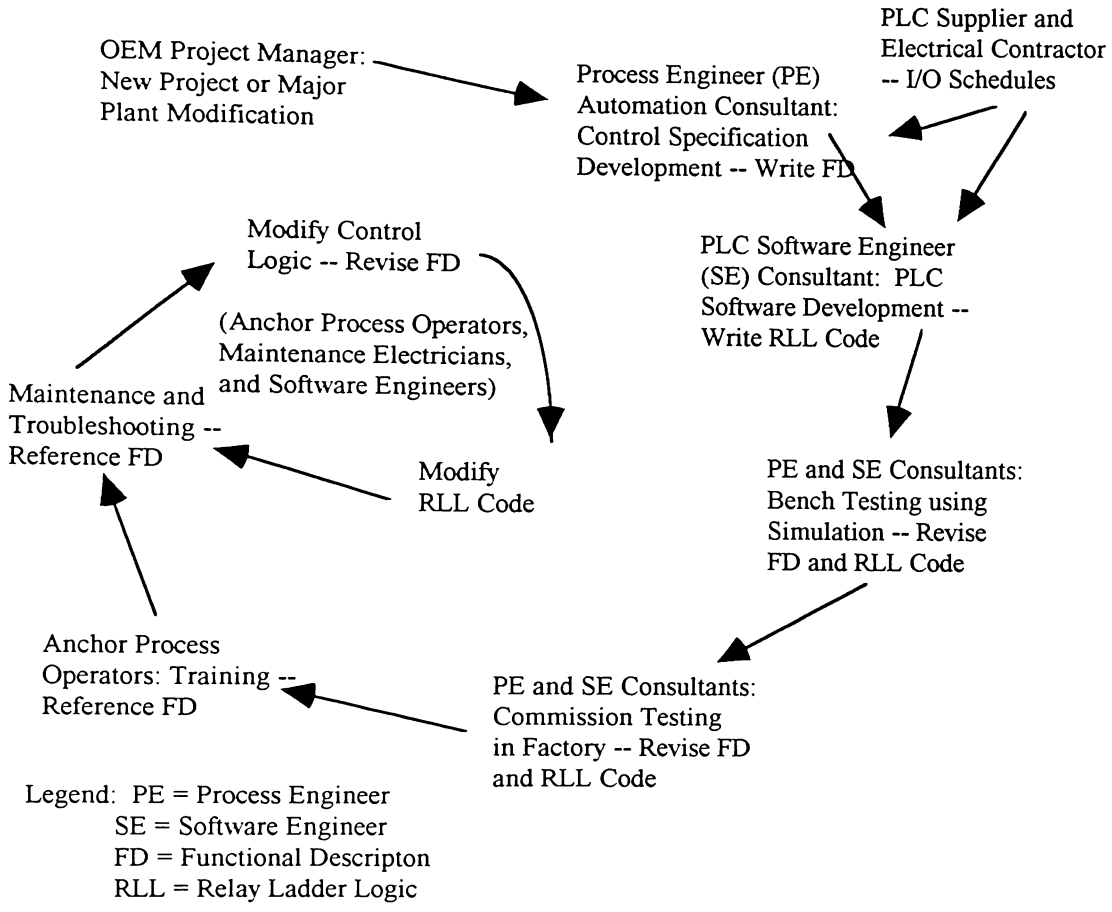


Figure 4.16. Idealised Life Cycle Showing Participants.

There are two primary products of these activities: the *functional description* (FD) of the control logic and the *PLC software*, consisting of relay ladder logic (RLL) code and the PLC data table. The functional description is written by the process engineering consultant, using a word processor, and serves as the requirements for the PLC control software. The FD is then manually translated into RLL code by the PLC software engineering consultant, using the PLC vendor's RLL editor. The software engineer also configures the PLC data table which specifies the I/O addresses between the PLC and factory devices and other information needed for programming the PLC. Information about the physical I/O comes from the PLC supplier and the electrical contractor.

Following translation of the FD into RLL, bench testing of both the FD logic and RLL code begins. In the bench testing phase the process and software engineering consultants work together to debug both the control logic (the FD) and the software

(the RLL). The RLL code runs in a PLC, and the plant is simulated using software. Any RLL coding problems or FD logic problems discovered at this point are repaired directly in the RLL, with annotations written on the hard-copy FD for later incorporation into the final FD by the process engineer. It is at this point that the FD and the RLL begin to diverge: The RLL code was no longer written solely from the FD specification. Practitioners report that during bench testing that 15%-20% of the FD is modified.

Following the bench testing phase, the debugging process moves to the factory, where testing continues in the plant, first using water and finally using product in the factory. This phase is called commissioning. Again, logic or coding problems are repaired directly to the RLL code and logic changes are made as annotations to the hard-copy FD, later to be incorporated into the final (commissioned) FD by the process engineer. Information from practice shows that up to 5% of the FD is modified during the commissioning testing.

At the completion of commissioning, and after a warranty period of up to one year, the control software and the hard copy FD are turned over to the Anchor personnel for maintenance. This hard-copy of the FD will continue to be used as an operations manual to guide factory floor staff in troubleshooting operational problems. The process engineering consultant's involvement with the FD document ends at this point.

During maintenance and troubleshooting, enhancements or maintenance changes are made to the factory. In the idealised life cycle, any required changes to the control logic should be designed and documented in the FD, then translated to RLL software in the PLC. However, in practice the changes are made directly to the RLL software in the operational PLC. The corresponding logic changes to the FD are (sometimes) hand written onto the hard-copy FD. Thus, the RLL and FD continue to diverge, and over time the FD may no longer reflect the actual operating control code. Information from practice indicates that after a period as short as eighteen months of operation, the FD is

typically no longer of value in understanding the PLC control software due to lack of maintenance of the FD.

The maintenance and troubleshooting phase in reality, as opposed to the idealised life cycle, is illustrated in Figure 4.17. The Xs indicate sub-phases that are rarely performed, and the double-headed arrow indicates that logic changes are made directly to the operational PLC control code, without modifying the FD before hand.

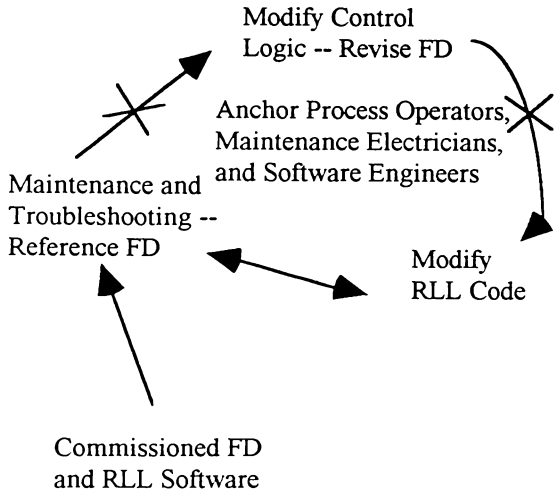


Figure 4.17. Maintenance and Troubleshooting Phase in Practice.

This environment reflects many of the same industry-wide problems in control programming for PLCs that were discussed in Chapter 1. In particular, the RLL code is poorly coupled to the design phase. The RLL is a manual translation of the control specification (FD), and as soon as testing begins, the RLL code diverges from the specification. Thus the FD, which is the specification of the RLL code, is not maintained and becomes progressively more useless as an operations manual, for troubleshooting, or for use as a basis for understanding the process logic and the RLL code in order to make future changes.

Another problem is that the structure and command language used to code the FD has never been standardised, neither at Anchor nor in the batch control industry. (The sample FD supplied to the author for this investigation was "representative" of a FD, but not a standard.) Typically, independent process automation engineering consultants are contracted to develop the control logic. Each engineer uses their own approach to

writing the functional description. Over the years, a generally accepted approach has been adopted, but the details of the structure and command language of the functional descriptions still varies between individuals. This places a burden on the software engineer who must adjust to the variations. In discussions with process and software engineers, the author found some fundamental misunderstandings in the interpretation of the function description. For example, in the absence of explicit rules, the process engineer assumed that multiple conditions in the preconditions portion of a step were related by logical "and" operations whereas the software engineer assumed that they were related by logical "ors".

A study of the effort in person-days for the development and testing phases of the life cycle revealed the statistics shown in Table 4.3.

Task	Effort (person months)	% Total Effort
Develop FD	12	26%
RLL Coding: Data Table and Messages	5.6	12%
RLL Coding: Logic	16.8	37%
Bench Testing	5.6	12%
Commission Testing	6	13%

Table 4.3. Effort In Life-Cycle Phases for Development.

The PLC RLL coding phase consists of development of the PLC data table, that shows the mapping of real plant I/O devices to PLC memory addresses; development of a message database, showing status and fault messages that are displayed on the man-machine interface (MMI) using numeric codes stored in the PLC; and finally the code for the logic in RLL. Of the total effort for the coding phase (22.4 person months), 5.6 person months, or 25% of the effort, are devoted to development of the data table and messages. Anchor software engineers report that on some projects, the development of the data table and messages can account for up to 40% of the PLC coding effort. Together the FD development and the coding phases account for three-fourths of the total development effort.

These statistics indicate that targeting the FD and PLC software development phases for automated assistance using CASE tools could have an impact on 75% of the effort in the life cycle prior to the maintenance phase. Furthermore, provision of automated assistance for maintenance of the FD was identified as a high priority by Anchor management.

4.8 Analysis of Computer-Based Tool Support in Current Life Cycle

Table 4.4 summarises the software tools used in the software development life cycle at Anchor.

Development Phase	Product	Practitioner	Software Tool(s)
Control Specification	FD	Process Engineer	Word Processor
PLC Software	RLL Control Code	Software Engineer	PLC Vendor's RLL Editor and Database Management Software for Data Table Development
Bench Testing	Revised FD and RLL Code	Process and Software Engineers	Proprietary Plant Simulation Software, Word Processor, and PLC Vendor's RLL Editor
Commission Testing	Revised FD and RLL Code	Process and Software Engineers	Word Processor and PLC Vendor's RLL Editor
Maintenance and Troubleshooting	Revised RLL Code	Anchor Staff	PLC Vendor's RLL Editor

Table 4.4. Software Tools Utilised in the PLC Software Life-Cycle.

The following is an analysis of the computer-based assistance in the current system for support for the domain and also for the dimensions of CASE technology functionality using the categories of production, coordination, and organisational support.

4.8.1 Domain Support

In Chapter 3 a definition for domain specific support in a CASE tool was developed based upon abstraction, specialisation, and the concept of cognitive distance. A domain specific tool utilises abstractions that are expressive, but at the same time reduces the

effort necessary to specialise the abstraction by minimising the variable portion of the abstraction and maximising the fixed and hidden parts. Minimising the effort to use the tool reduces the cognitive distance of the tool, which is a primary goal of domain specific methods.

The software tools listed in Table 4.4 are all generic and domain independent. In the control logic development phase of the life cycle, the word processor is highly generic. It's abstraction is natural language text. Thus, the variable portion of the abstraction is large requiring extensive effort to specialise it to represent the structure and language of the FD, thus giving it a large cognitive distance. The PLC vendor's RLL editor is also generic in that it is targeted toward any potential RLL programming application. It provides no automated support for mapping from the abstraction specification (the FD) to the abstraction realisation (the RLL code).

A problem with using generic tools is that users must expend considerable effort to specialise them, beginning from scratch (or first principles) in order to apply them to a particular application domain. These tools do not act as methodology companions by incorporating domain specific standards, languages, or other helps. The advantage of generic tools is their large domain width -- by their generality, they may be applied in a large variety of situations, but at the cost of requiring extensive effort on the part of the user to specialise them (their cognitive distance is great).

The implications of the lack of domain support in the current software tools are further examined in terms of the dimensions of CASE tool functionality.

The following three sections analyse the functionality of tools in the current system using the three dimensions of CASE tool functionality that were identified in Chapter 3: Production Technology, Coordination Technology, and Organisational Technology. This section is an analysis of the functionality of the tools in the first category, production, which is broken into the sub-categories of representation, analysis, and transformation.

4.8.2 Analysis of Current Tool Functionality: Production Technology

Representation functionality enables the user to define, describe or change a definition or description of an object, relationship, or process (Henderson, 1990). It has been shown in this chapter that a representative structure and command language for specification of batch control exists in both the literature and at Anchor and, moreover, the methods in use at Anchor are quite similar to those in the literature. A domain specific tool to support the representation of batch control logic could minimise cognitive distance by embedding this methodology into the tool. This is the essence of a CASE methodology companion, which is designed to structure the software development process according to the steps and rules of a particular methodology. For example a methodology companion provides information about the methodology by means of menu choices and also enforces completeness and conformance to standards (McClure, 1989). In this way, the methodology companion reduces the effort (cognitive distance) to enter a design specification by minimising the amount of specialisation required to use the tool.

The word processor is the antithesis of a methodology companion. For example, the word processor does not "know" that steps are composed of preconditions, operates, and checks and thus cannot prompt the user for this information nor can it enforce that the information is present. Also, the word processor does not "know," for example, that a pump can be started or stopped, whereas a valve can be energised or deenergised. A methodology companion could supply this sort of information in lists or menus and also enforce the correct structure and terminology.

Another problem due to the need to specialise the word processor is that users indicate that they spend significant amounts of time working with the format and structure of the FD document rather than the logic of the process. They spend time in working with fonts, column layouts, etc., which could be built into a domain-specific tool. This difficulty carries over into the maintenance phase, where (as will be noted below under Cooperative Technology) the FDs are not maintained. The difficulty in

using the word processor on the part of plant operators impedes attempts to maintain the FD.

To reuse a FD, the engineer uses the word processor's cut/copy/paste and search/replace capabilities. Although a powerful feature, the word processor does not aid in copying and pasting identifiable portions of a FD. For example, to copy an entire selection, the user must select all of the steps by dragging the mouse pointer over the entire text of the selection. This is because the word processor treats the document as free-form text and not as specific, identifiable sections. It would be simpler if, after the user clicks copy, the CASE tool would allow the user to specify the selection's name, and the CASE tool would supply the functionality to obtain all of the steps for the selection. Similarly, with search/replace the word processor operates on the entire document, whereas the designer often desires to restrict a search/replace operation to a given selection. A domain-specific CASE tool could provide this capability.

Analysis functionality enables the user to explore, simulate, or evaluate alternate representations or models of objects, relationships, or processes. Examples include the ability to simulate a design or ask "what if" questions (Henderson, 1990). The use of generic office automation tools, such as the word processor, does not provide support for analysis of the functional description. For example, as a design aid, the process engineer sometimes produces cross reference listings, showing where in the FD various plant items have been referenced. In the old environment, these reports are produced manually using search, copy, and paste word processor commands or must be delayed until after the RLL code has been written where the PLC vendor's RLL editor can supply some of these functions on the RLL code itself. Furthermore, these generic tools do not support simulation (model-based analysis) of the logic in the FD. It is a well established principle in software engineering that the later in the life cycle that problems are discovered, the more they cost to repair. Currently, testing of the FD logic is delayed until after it has been completely translated into PLC RLL code. Problems found during testing, after being corrected in the PLC, are "reverse engineered" back into changes to the FD! To summarise, using the word processor, the

production of any sorts of analysis or inspection reports is a time consuming manual process involving search and copy operations. There is no capability for simulating or testing the FD's logic from the FD specification in the current environment.

In the testing and maintenance phases, the current tools provided some assistance in the sub-category of analysis. The PLC vendor's RLL editor provides some reports such as cross references to aid the software engineer's task of translating the FD to the RLL code. In the testing phase, the current environment includes a plant simulator for testing of completed RLL code (and the FD logic) with the PLC. Although the availability of the RLL editor's reports and the plant simulator are significant and important features, they can be applied only after the control logic has been transformed into RLL code.

Transformation support is functionality that executes a significant planning or design task, thereby replacing or substituting for a human designer/planner (Henderson, 1990). A lack of tool integration during transformation of the FD to RLL is evident. The word processor used to produce and maintain the FD is not integrated with the tools (RLL editor and possibly a database system to help design the PLC data table) used to produce the PLC control code. The FD is printed as a paper document and handed off to the software engineer who manually translates the logic specified in the FD's command language into RLL code using the PLC vendor's proprietary RLL programming environment. Similarly, in the bench testing and commission testing phases, any problems found in the control logic must be corrected twice: in the FD and in the RLL code. Thus, the old environment provides no automated support in this area.

The manual programming process is complicated by the lack of FD standardisation and thus software engineers must cope with different FD languages and styles. Adapting to different FDs requires time and effort, and practitioners also report that this introduces ambiguities in trying to understand the meaning of various engineer's FDs.

A phenomena that occurs during the translation is that the software engineer supplies code for requirements that are missing from the FD. The FD does not specify detailed requirements for reporting, operator interface-related code, background fault checking, overall code structure, and other non-control related functionality. For example, recall the force on and force off bits coded in the RLL code (Section 4.6). The FD does not indicate requirements for this code. The software engineers supply this code based upon experience or interactions with other developers in the project.

A weakness related to translation (but more appropriately described under the Organisational Technology category) is the fact that the RLL language is not standardised, nor are RLL programming environments standardised. This means RLL programming skills are not completely portable from one PLC to another. This requires either training of software engineers for each make of PLC or specialisation of individual engineers for each make of PLC.

4.8.3 Analysis of Current Tool Functionality: Coordination Technology

This section analyses the functionality of the current tools in terms of Coordination Technology, which is divided into the areas of control and cooperative functionality.

Control functionality enables the user to plan for and enforce rules, policies, or priorities that will govern or restrict the activities of team members during the planning or design process (Henderson, 1990). As noted in the above section, the word processor does not act as a methodology companion by enforcing methodology standards nor does it provide domain-specific help to guide the user. It therefore does not aid in controlling the development of the FD to a standard.

A primary weakness in the existing life cycle is the difficulty of maintaining the FD document during the maintenance phase. During maintenance, changes in the manufacturing plant necessitate changes to the RLL control software. The methodology for making these changes is not the same as that followed during initial development. That is, the specification for the change is not first written in the FD, then translated to RLL and then loaded into the PLC for testing and commissioning.

Instead, operational changes are made directly to the RLL code and inserted into the PLC on-line, because in a production system the PLC cannot be taken off-line for reprogramming. Later, the corresponding change in the FD specification may (or may not) be hand-written into the hard-copy FD. Moreover, manual annotations may be inaccurate and may not be inserted into all copies of the FD. Finally, as was noted above in the Production Technology discussion, updating the FD using the word processor is not done because the maintenance staff is unfamiliar with the use of the word processor and they have hard-copy FDs only.

Cooperative functionality enables the user to exchange information with another individual for the purpose of influencing the concept, process or product of the planning/design team (Henderson, 1990). Since the FD language has never been standardised different process engineers find it difficult to understand each other's FDs, software engineers find it difficult to adjust to differing FDs, and plant operations staff find some FDs difficult to understand. The lack of FD standardisation thus adversely affects productivity of individuals, it limits the capacity of individuals to work together on projects, and it stifles information flow. For example, programmers and operators find it difficult to communicate ideas to the FD designer when they are unfamiliar with the particular FD style. A consistent FD should facilitate interaction of individuals during the FD development process which in turn should translate into quality and productivity improvements.

Winograd and Flores note that many systems designed by computer professionals are intended to facilitate the activity of an individual working alone (Winograd, 1986). Examples cited include word processors, filing systems, and program creation aids. Although useful, these tools leave out the essential dimension of collective work. This state of tool-support describes that at Anchor when this investigation was initiated: throughout the software development life cycle, individuals use tools designed to facilitate an individual activity, not to support coordination or communication across the life cycle.

Winograd also notes that in most work environments the coordination of action is of central importance. In this view, the coordination dimension permeates every realm of coordinated activity, whether it be computer programming, medical care, or simply selling shoes. Although the details differ from setting to setting, computer systems can be used to assist in the coordination of an organisation's activities. The current tools offer little assistance in this regard.

4.8.4 Analysis of Current Tool Functionality: Organisational Technology

Organisational Technology is the third area of CASE tool functionality. It is composed of the sub-categories infrastructure and support.

Infrastructure includes functionality and standards that enable portability of skills, knowledge, procedures, or methods across planning or design processes (Henderson, 1990).

The lack of standardisation of FDs makes it difficult for FD users, such as software engineers and plant operators, to move from one project (or plant) to another. They report that it is difficult to adapt to and understand one FD approach when they are used to another. Enforcement of a standard FD by the development tools should contribute to the capability of individuals to move between projects and plants. Even seemingly simple changes, such as the way that the FD is printed on the page, can pose adjustment difficulties for plant operations personnel.

The lack of a standardised RLL language (and programming environment) impedes the ability of software engineers to move from one project to another where different PLCs are used. Automatic code generation (ie., automated mapping of the abstraction specification to the abstraction realisation) of RLL from the FD may enable skill portability of software engineers.

Support functionality helps an individual user understand and use the development environment effectively. Examples include the use of domain knowledge or proactive help systems (Henderson, 1990). As was described in the sections on representation and control, generic software tools are domain independent and thus do not assist the

user with the development methodology by offering domain-specific helps. The word processor and the RLL programming environment are all generic tools in the sense that they do not incorporate domain-related rules or languages. Another problem is that the current set of generic tools (word processor, data base management system for the equipment data base, and the RLL programming environment) are not integrated thus requiring manual effort to translate information from one representation, FD, to another, RLL.

4.9 Summary

A domain analysis was performed to document current practice in the field of batch control software development. A review of the literature revealed that there currently is no generally accepted standard for the structure and language for specification of control logic. However, work has been made in the area, and a representative structure and language for batch control software has been proposed in the literature. The structure reveals a hierarchical structure and a typical text-based control command language.

A study of the methods used at Anchor Products also revealed a lack of standardisation in the structure and language for control system specification. The control logic is specified in a text-based document called the Functional Description. A study of a representative FD revealed that Anchor's methodology is quite similar to the representative methods from the literature. The hierarchical structure and the text-based control command language used in Anchor's FD correspond closely with the results of the literature survey, with differences in terminology and the fact that steps in Anchor's method typically operate concurrently instead of strictly sequentially.

Analysis of the tools used in the current environment showed that the software tools are domain independent (or generic). The software tools use generic abstractions that require significant effort to specialise in order to produce the FD and the PLC software. Thus their cognitive distance is large, requiring significant intellectual effort on the part of the developers in order to take the software system from one development phase to

another. In addition, the tools in the current environment were developed to facilitate the activities of an individual working alone, and generally leave out the essential dimension of coordination and collective work.

The results of the domain analysis and consultations with Anchor personnel reveal that there is significant opportunity to improve the old control code development environment by using appropriate CASE technology that:

- Acts as a methodology companion by embedding methodology information in the tools, thus enforcing a FD standard and aiding the development process.
- Uses abstractions that require minimal specialisation to minimise cognitive distance, thus providing an easier to learn and use tool.
- Automatically maps the specification (FD) to the realisation (RLL code) (automatic code generation) and thus substitutes automation for manual effort which encourages maintenance of the FD, and also provides better tool integration.
- Provides model-based analysis of the FD to support testing logic earlier in the life cycle.
- Provides other analysis reports of the FD to assist the development process.

Table 4.5 shows the problems identified in the domain along with the ideas from the theoretical framework (Chapter 3) that will be applied to address the problems. The dimensions of CASE tool functionality are used to categorise the problems.

CASE tool functionality dimension	Domain problems	Theoretical concept to address th problem
Production Support		
Representation	Generic tools do not embed domain-specific methodology support. Significant effort is expended to specialise the tools for use in the domain.	Domain specific methodology assistants require less specialisation.
Analysis	No automated support for generation of analysis reports from the FD Testing of logic deferred until after PLC coding	Generation of analysis reports from the FD by CASE tools Model-based analysis of the FD
Transformation	Increasing complexity of PLC software PLC software is not portable Specialists required for each PLC type	Automatic generation of PLC code from the FD
Coordination Support		
Coordination	Effective communication is impeded by the lack of a FD standard Tools facilitate individual activities, not collective work	Standard enforced by CASE tools CASE tool designed to support the organisation, as well as individuals
Control	FDs are not maintained	Design maintenance and automatic code generation
Organisational Support		
Infrastructure	Lack of FD and RLL standards impedes portability of skills	Standard enforced by CASE tools Automatic generation of RLL code
Support	Generic tools lack domain assistance, are difficult to use, and are integrated weakly	Methodology assistants provide domain specific help and guidance Integrated CASE technology

Table 4.5. Problems in the Current Environment and Concepts to Address Them.

The next step in the investigation was to design a CASE tool based upon the framework of ideas presented thus far. Before commencing the design process, several meetings were held with representatives of Anchor to discuss the problems to be addressed and possible solutions. This process (which represented the first action

research cycle) resulted in the articulation of the roles of the researcher and the client, and also of an approach to guide the design of a domain-specific CASE tool. These activities are the topic of the next chapter.

Chapter 5 A Method to Guide the Design of Domain-Specific CASE Tools

This chapter describes the initial intervention by the author into the client organisation and the interaction between these two parties that culminated in the decision to proceed with the design of the prototype CASE methodology companion. It represents the first major cycle in the action research approach. A major result of this phase of work is the articulation of a method to guide the design of domain-specific CASE tools which is subsequently applied (in Chapter 6) to produce a prototype CASE methodology companion for the development of batch-control manufacturing software.

5.1 Point of Intervention and Definition of Roles

Recall from Chapter 2 that efforts to bring about change in an organisation typically begin with either the people, the technology, the structure, or the task (Leavitt, 1965). At an initial meeting between the researcher and managers from Anchor, agreement was reached that the point of intervention would be to attempt to influence the PLC software development life cycle by introducing new technology. The client organisation was interested in standardising the functional descriptions and also in making general improvements in the process of developing control code. The author was interested in investigating the features that constitute an effective CASE tool and the potential and limitations of CASE technology on the software development life cycle for developing manufacturing control software.

As was indicated in Chapter 4, the state of tool support in the client organisation when the researcher began the study was one of a suite of generic tools, such as a word processor, data base system, and RLL programming editor that were designed to facilitate the activity of an individual working alone. This situation could be labelled "islands of automation." This point of view on the part of the client organisation was further illustrated by their initial proposal for improving the technology.

Early in the study, engineers in the client organisation proposed the development of a FD compiler. The FD compiler would accept a standardised FD in text format, written using a word processor, validate for it syntactic conformance, and then possibly translate it into PLC code. The existing tools and methods would remain intact except for the introduction of the FD compiler, which would be used to enforce standardisation and to partially automate the software engineer's coding task.

In contrast, the author was interested in a making more substantial changes to the technology used in the life cycle. In essence, the author was interested in merging the islands of automation and to then assess the impact of this change on the organisation. The author's objective was to determine the potential for CASE technology to integrate and automate the activities of the process and software engineers and to assess the effectiveness of this approach.

Thus, early in the study, the author endeavoured to introduce managers in the client organisation to previous work in the development of automatic code generators for PLCs, and in particular to introduce them to the potential for CASE tools to provide integrated support for the entire life cycle. This process of education led them into agreement with the author's perspective. Furthermore, management agreed that it should be the researcher's (author's) role to research, propose, and design an appropriate CASE tool and that the client organisation would provide access to information and to key people. Management also agreed to review proposals and progress of the work and to help to assess the potential of the resulting tool.

This initial action-research cycle served to broaden the perspective of individuals in the client organisation and began to change their views on the role of software tools in the life cycle. It served to define the roles of the researcher and the client organisation, and freed the researcher to pursue his ideas for intervention and change, namely the design and introduction of a CASE methodology companion to support the client organisation's software development life cycle.

A third result of the initial collaboration was the development of an approach to guide the design of domain-specific CASE tools. As the author conducted the literature review and the domain analysis, reported in Chapters 3 and 4, and as he worked with the client organisation to define an appropriate specification language for batch manufacturing, he found that he was eliminating languages from consideration that required extensive amounts of specialisation in order that they be useable for this particular domain. This is why graphical languages such as Grafcet and Petri-nets were deemed undesirable by the engineers at Anchor as well as the fact they were familiar with an unsuccessful project at another dairy that attempted to use Grafcet. This experience led the author to articulate the following approach to guide the design of domain specific CASE tools.

5.2 CASE Tool Development Methodology

This section outlines an approach to guide the development of domain-specific CASE tools. The approach was used to produce the prototype CASE methodology companion described in Chapter 6.

5.2.1 Approach

In Chapter 3 the concept of domain-specificity was explored in terms of abstraction, specialisation, and cognitive distance (Krueger, 1992). Also a method for developing application generators, which are highly domain-specific, was identified (Cleaveland, 1988). These results are combined with the procedures of domain analysis to yield the following methodology for development of domain-specific CASE tools.

1. Identify the domain by performing domain analysis. Identify methodologies, models (abstractions), and methods that are appropriate to the domain, including appropriate abstraction specifications and abstraction realisations.

2. Decide what aspects of the domain should be included in the CASE tool. The domain width should be only as wide as is necessary to accommodate user needs.

3. Identify the abstraction (or model) presented to the user. Choose a specification language that is acceptable to the user, is succinct and expressive. The 80/20 rule,

where everything in a domain falls into a clean computational model, sometimes applies (Levy, 1986). In this case provision must be made so that the 20% can be programmed directly into the implementation technology.

4. Decide what aspects of the abstraction are under the control of the user (variable part) and which parts are fixed. The objective is to minimise cognitive distance.

5. Define the way the user specifies the system under development. Options include textual specification languages, templates, graphical diagrams, interactive menu-driven dialogue, and structure-oriented editing.

6. Define the products, or realisation, of the CASE tool. Researchers suggest that CASE tools should automatically generate code from the specification and should produce reports to aid in the analysis and execution of the specification.

7. Implement the CASE tool.

Each of these steps will now be considered.

5.2.2 Perform a Domain Analysis

The purpose of the domain analysis is to identify abstraction specifications and realisations appropriate to the domain. Chapters 3 and 4 serve this purpose.

Chapter 3 identified the three abstractions (or models) utilised by most real-time methodologies. These are the functional, behavioural, and structural views. Commercially available CASE tools for real-time control incorporate these three abstractions but may use alternative specification methods. Specification methods are describe in section 5.2.4.

Abstractions particular to the specification of PLC control software were also identified. The most common abstractions are a list of operations that include pre- and post-conditions. Other possible models include Petri nets (or the related Grafcet approach) and finite state machines.

An analysis of the literature revealed a representative structure and command language for the specification of batch control logic. This structure and language is

quite similar to that in use at the industry partner (Anchor Products) which was described in Chapter 4. The structure is hierarchical and the control logic is specified using steps that include preconditions, operates, and fault checks.

The most common realisation of batch control software in industry takes the form of relay ladder logic code for execution in PLCs. Commercially available CASE tools for real-time control do not generate RLL, but typically generate C or Ada code. Domain specific code generators for PLCs have been investigated that generate RLL or its boolean equivalent. These generators are typically based on a specification as a list of operations and associated conditions.

5.2.3 Decide What Aspects of the Domain to Include in the CASE Tool

The CASE tool should support batch control software for PLCs. Discussions with practitioners revealed that from 10% to 20% of the code in the operational control system was not specified in the FD. Code outside the domain includes that necessary for reporting, background checking of plant devices, and higher-level "integration" code that coordinates all of the selections in the PLC. A decision was made by the author to include support for the batch control logic in the CASE tool and to make provisions for coding the other 20% either through templates in the code generation module (described below) or directly using the traditional RLL coding methods.

5.2.4 Identify the Abstraction Presented to the User

The abstraction that is most efficient and provides a small cognitive distance is the batch control software structural model and command language identified in the literature and in practice at Anchor. This abstraction expresses the information needed for the control and is at the same time narrow thus requiring minimal specialisation. The following is a justification for this decision.

Specification abstractions used in the major real-time control methodologies include data flow methods for the functional view, finite state machines for the behavioural view, and individualised structure diagrams and entity-relationship diagrams for the structural view. Although batch process control could be specified using these views,

they all require effort to learn and specialise. All of the methods described above are more general than is required to support the domain of batch control logic and would require extensive specialisation on the part of the user to recreate the batch model. Support for specification of a structural view is in fact unnecessary because the structural model will be the same for every batch project. Thus, the batch structural model can be embedded into the CASE tool. Specification for a functional view is also unnecessary. The functional (or process) view depicts data flows and the transformation of data as it flows through a system (Hatley, 1987). In batch control processing, there is minimal data flow between processes -- the primary activity is control (or behaviour).

Abstractions used in past real-time control methods for the behavioural specification include finite state machines and Petri nets or Grafcet. The author considered advocating the use of Grafcet as an alternative specification language to the text-based control language primarily because Grafcet is an emerging industry standard and because several researchers have advocated the use of graphical specification languages (IEC, 1991; Stear, 1992; Harel, 1992; Roberts, 1990). However, this proposal was abandoned because of the way that the process engineers conceive of the steps of a manufacturing process: They are all independent and may be executing in parallel, but yet there is typically a strong dependency between steps. For example, a precondition of step 5 could be that step 4 be active. A precondition of step 4 is that step 3 is active. However, steps 3, 4 and 5 will all eventually become active concurrently. This type of dependency can be specified in Grafcet, but requires that each step be written as individual diagrams and the graphical advantages of Grafcet in showing flow of control disappear. Classic finite state diagrams also suffer this disadvantage. Another reason for rejection of Grafcet is that it is meant as an overall structuring mechanism to be utilised with another lower level language, such as RLL or structured text, so a lower level control language must still be utilised.

Another notable difficulty with Grafcet is that it can be awkward to code for exceptions, as the network begins to complicate with numerous and crossing lines (David, 1992). (Modifications have been proposed to address this problem (David, 1992).) Finally, as was mentioned above, the sponsoring organisation was familiar with a negative experience in the use of Grafcet at a competing organisation.

An alternative graphical notation is that developed by Roberts using product flow and behavioural diagramming techniques resembling data flow and finite state diagrams (Roberts, 1990). This research demonstrated the feasibility of translating that graphic notation to RLL. However, for this investigation, this language presents a domain width that is broad, requiring the user to specialise it to the batch methodology.

Specification abstractions used by PLC code generators typically utilise a data base containing a list of operations and associated pre- and post-conditions. These operations specify the behaviour view, but include neither a structural (eg. hierarchical decomposition) model.

Thus, the author and the engineers at Anchor agreed that the most appropriate abstraction in terms of minimising cognitive distance and providing an efficient abstraction for this investigation is Anchor's FD structure and command language, but one refined into a single standard.

5.2.5 Decide What Aspects of the Abstraction that are Under the Control of the User

A CASE tool that embeds the batch structural model into the fixed part of the abstraction specification, but allows specification of any number of selections, steps, preconditions, operates, and checks and supports an acceptable form of the representative batch control language, will support the desired domain while minimising the cognitive distance of the tool.

Since the terminology used in the representative batch control language follows a general structure, but the terminology used within the structure is not standardised, a design decision was made by the author to allow the user to specialise the particular

terminology used by the CASE tool. For example, one engineer may want to use the command "OPEN VALVE1" while another may prefer the command "ENERGISE VALVE1".

In the same vein, users should be able to specify the naming conventions for plant items and the meaning of those names. For example, VP could refer to a two position valve with feedback, OP to indicator lamps that have no feedback, and P to pumps with a single feedback signal. Furthermore, the user should be able to define rules relating terminology to types of plant items so that the CASE tool can guide the user as to terminology that is appropriate for particular plant items. For example, it is valid to use OPEN with valves, START with pumps, and ENERGISE with indicator lights. Rather than build these rules into the CASE tool, they should be user configurable.

To provide flexibility in the mapping of the command language to the RLL code, the user will be provided with the capability to define the I/O addresses of plant devices and the mapping between terminology (such as OPEN or START) and corresponding RLL code. Also, because all of the RLL code in an operational control system is not specified by the FD, provision is made for the user to code this directly.

5.2.6 Define the Way the User Specifies the System Under Development

Since the language to be adopted is the representative batch control language, a forms-based visual interface is appropriate. Forms can be used to guide the user, provide pull-down help menus and lists, and minimise the amount of data entry required by the user. The content of the forms was derived from the model sample FD, with some extensions from Grafcet, for time-limited, time-delayed, conditional, and one-shot operations. The interface is discussed further in Chapter 6.

5.2.7 Define the Products of the CASE Tool

The CASE tool will generate PLC code corresponding to the control specification, documentation and analysis reports, and model-based analysis reports. The most appropriate PLC code is relay ladder logic. Since it is anticipated that the tool may automatically generate 80% of the PLC code, the code will need to be a form that is

extensible and understandable by the software engineers. RLL code fits this requirement. Documentation and analysis reports include printed copies of the FD, as well as more sophisticated reports such as summaries and cross references. These reports are intended to not only automate production of reports previously produced manually but also to produce reports (such as cross references) that were done rarely because of a lack of tool support to generate them. Model-based analysis may will include execution of the specification using simulation to support some testing of the batch logic prior to the coding phase.

5.2.8 Implement the CASE Tool

The detailed design and implementation of the tool is the topic of the remainder of the next chapter. At the request of the sponsoring organisation, the prototype CASE tool was implemented as a Microsoft Windows™ application. Microsoft Visual C++™ was used as the implementation language because of its support for Windows programming and the desire to use object-oriented implementation methods.

The goal in implementing the CASE tool to use it as the research vehicle to address the problems in the domain and to collect information to verify and validate the methodology and framework on which the tools are based and to evaluate its impact on the client organisation.

5.2.9 Summary -- CASE Tool Development Methodology

The application of the methodology for development of a domain-specific CASE tool has led to the following conclusions. The CASE tool will address the domain of batch control software for PLCs. The abstraction presented to the user will be standardised and based upon Anchor's FD structure and command language, as opposed to the more general real-time methods which have a large cognitive distance or that of past PLC automatic code generators which do not incorporate a hierarchical structure. The user will be able to control the behavioural aspects of the logic, but the structural view will be embedded in the CASE tool. Support for a functional view is not required in this domain. The user will also be able to configure the particular

terminology used in the command language and associated rules describing how the terms are utilised in the language. Support will also be provided for specification of plant I/O addresses for individual plant items. Products of the CASE tool will include documentation, analysis reports, simulation of the logic, and generation of RLL code corresponding to the control specification. Provision must be made for the user to define mappings between symbolic names, such as VALVE2, and PLC I/O addresses, between terminology and RLL code fragments, and to allow the user to directly code software that is not covered by the batch language. The user interface will be a menu-and-forms-based interface implemented running on Microsoft Windows.

5.3 Potential of User Collaboration in the CASE Tool Design

Although the author was responsible for the design of the CASE tool, close collaboration with people in the client organisation was nonetheless necessary, both to perform the domain analysis, and also to obtain feedback concerning design alternatives to insure that necessary functionality was incorporated into the CASE tool's design.

Winograd and Boguslaw point out the system designers wield significant power because of the potential effect that a computer system may have on an organisation (Winograd, 1986). Bougslaw states that:

> A designer of systems, who has the de facto prerogative to specify the range of phenomena that his system will distinguish, clearly is in possession of enormous degrees of power.... It is by no means necessary that this power be formalised through the allocation of specific authority.... It is in this sense that computer programmers, the designers of computer equipment, and the developers of computer languages possess power. To the extent that decisions made by each of these participants in the design process serve to reduce, limit, or totally eliminate action alternatives, they are applying force and wielding power in the precise sociological meaning of these terms (From Winograd, 1986).

Winograd goes on to note that designers and programmers are therefore invisibly empowered but often they may have an overall orientation not shared by others within the organisation. For example, computer professionals tend to place a high emphasis on efficiency but to devalue human factors. The resulting system design can erode the effectiveness of the organisation while concentrating on its efficiency.

The author, being responsible for the design of the CASE methodology companion, endeavoured to work with software engineers, process engineers, and Anchor management so that appropriate design decisions would be made in the development of the prototype tool. Status reports and demonstrations of preliminary versions of the CASE tool were presented at various points during its development in order to help the author learn and grow in his understanding of the domain and also to obtain suggestions from both managers and engineers about various features of the system. By working with the potential users of the system in this way it was anticipated that appropriate design decisions would be made and that this action research process would serve to educate all individuals about issues in the control code development life cycle and the potential of CASE tools to improve it.

5.4 Research Proposition Elaborated

It is a supposition of this investigation that a key reason that past CASE tools have not been effective in the specification phase (Potts, 1993) and that their impact on productivity and quality has been inconclusive (Henderson, 1990) is due to their large cognitive distance. Their cognitive distance is large because they employ generalised abstractions (targeted to a large domain width) which require excessive effort to learn and specialise to a particular domain. Domain specific CASE tools employ abstractions that are targeted toward a particular domain and thus require less effort to learn and specialise, and therefore may lead to better acceptance and effectiveness.

The introduction of CASE technology into the organisation should influence the life cycle by improving the production, coordination, and organisational technology and also affect the structural and people variables of the organisation. The CASE tool can be considered to be effective if the client organisation finds that the CASE technology will put it in a better position as compared to its current methods.

The research proposition can now be elaborated by deriving propositions from the framework of ideas articulated thus far. This is in keeping with the assessment approach (Section 2.5) which requires the statement of empirically disconfirmable

propositions that can be organised into theory and falsified by practitioners (Argyris, 1985; Yin, 1989). The hypothesis to be tested is that computer aided software engineering (CASE) tools that are domain specific, incorporate model-based analysis and automatic code generation, and incorporate an appropriate methodology and standards will be effective in supporting the software development life cycle for manufacturing control software for programmable logic controllers (PLCs). Effectiveness of the CASE tools should be indicated by:

- Improvements in production: The CASE tools will contribute to the capacity of individuals to accomplish development tasks with fewer resources. Productivity of individuals and the quality of their work will improve.
- Improvements in coordination: The CASE tools will contribute to control and maintenance of the design specification. The CASE tools will improve efficiency and effectiveness of individuals by enhancing their capacity to exchange information with others for the purpose of affecting the concept, process, or product of the development life cycle.
- Improvements in organisational technology: In comparison with the generic tools, a domain specific CASE tool will be easier to learn and use. The enforcement of standards will enable portability of skills, knowledge, procedures, or methods across the life cycle.
- Changes in people, structural, and task variables: The introduction of CASE technology will influence the roles or relationships of people and the tasks that they perform.

To test these propositions, a prototype CASE tool that incorporates the features described above was developed and introduced to practitioners. This is consistent with the action research approach which calls for the generation of an infrastructure as the key mechanism for addressing the problem situation and for generating new knowledge. This notion is also supported by Basili's conclusion that only improved production processes will result in higher quality software.

5.5 Summary

The first cycle of the action research approach and its results were presented in this chapter. This included the initial intervention, the definition of the roles of the researcher and the client organisation, and the identification of an approach to guide the development of domain-specific CASE tools.

The role definition freed the author to pursue the design of an appropriate CASE tool, and led the client organisation to agree to serve as a laboratory to provide information, feedback on progress, and a site to assess the potential of the CASE tool.

This early intervention served to expand view of software tools previously held by managers in the organisation from one of the use of individual productivity tools to one of using CASE tools to help to integrate the life cycle.

The approach to guide the development of a domain specific CASE tool serves to guide design decisions that are intended to produce a tool that minimises cognitive distance and is acceptable to the user community. The next chapter describes the design and implementation of the prototype CASE methodology companion developed for the investigation.

Chapter 6 A CASE Methodology Companion to Support Development of Batch Control Software

The software tools in the current development environment at Anchor Products highlight many of the problems found in PLC software development in general. Moreover, CASE tools have not been adopted at Anchor Products, and generic productivity tools have been utilised in their place. It is a hypothesis of this investigation that many of the problems can be addressed by introducing a methodology companion in place of these isolated productivity tools. Based upon the results of the literature review, the author hypothesised that incorporating the concepts of domain-specific methods, automatic code generation, model-based analysis, and an appropriate methodology and standards into the methodology companion would yield a tool that will address the problems identified in the development of batch control software. This chapter describes the detailed design of the CASE tool and presents an overview of its implementation.

6.1 CASE Tool Architecture

The generic CASE tool architecture (Figure 3.1) is used as a foundation for the prototype methodology companion for this investigation. However, the capability to specialise the batch command language, plant item naming conventions, and to define relationships between the language and manufacturing devices is added, as suggested by Robert's proposed environment (Figure 3.5). Figure 6.1 illustrates this architecture.

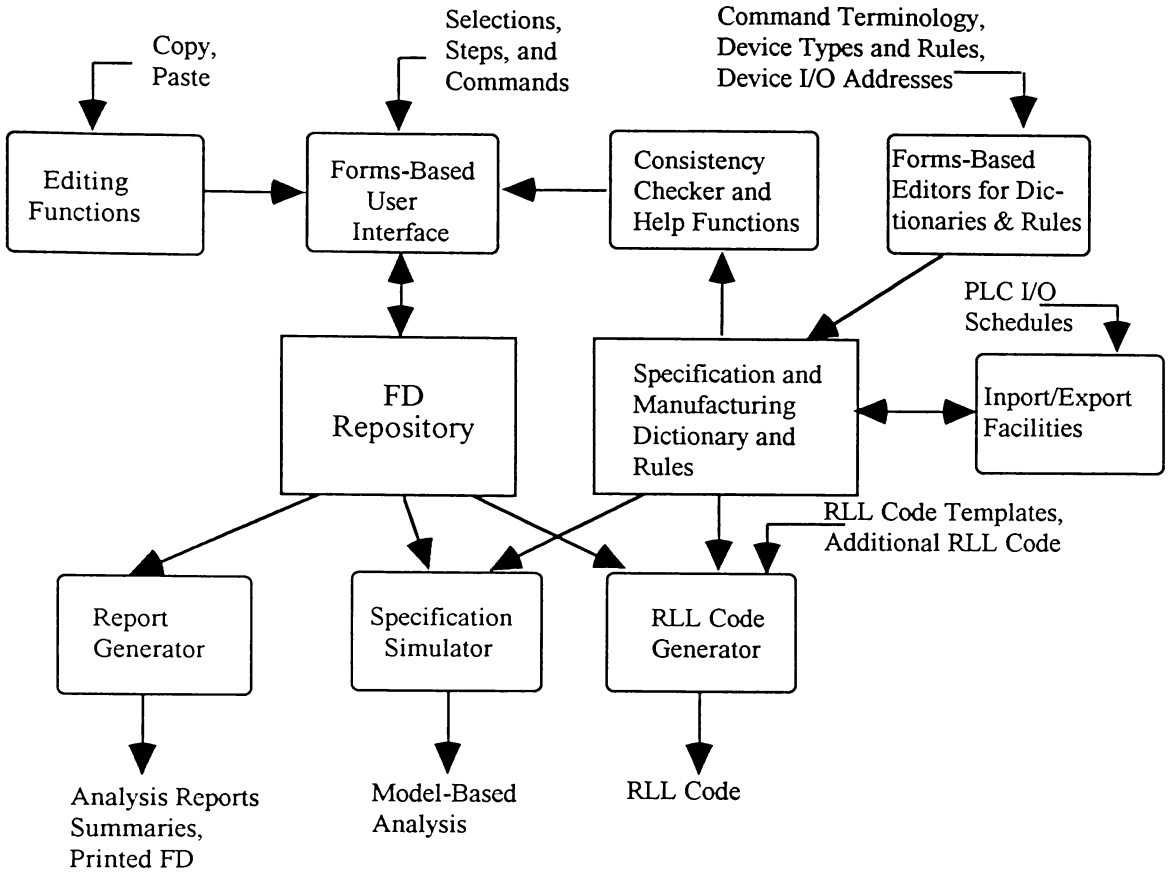


Figure 6.1. Domain-Specific CASE Tool Architecture (FDA).

The CASE tool is dubbed the Functional Description Assistant (FDA) in keeping with its aim of acting as a methodology companion for producing FDs and associated RLL code.

The following sections describe the implementation of the CASE tool.

6.2 CASE Tool Abstractions

Recall from Section 3.4 that most methods for real-time control development utilise the following three abstractions:

- the *functional* model, showing the relationship between the functions carried out by a system on the stored data.
- the *structural* (or *data*) model which deals with both system architecture and with the stored data modelling.

- the *behavioural* model, depicting the response of the system to events, as influenced by the passage of time and the history of previous events.

For batch manufacturing control, the structural model, based upon the representative hierarchical structure, can be embedded into CASE tool as a fixed abstraction. The need for a functional abstraction is minimal to non-existent, and is thus not necessary. Thus, only the behavioural model remains as a candidate for the variable part of the abstraction. The structural and behavioural abstractions are discussed below.

6.2.1 Structural Model -- the Fixed Abstraction

Based upon the analysis of the sample model FD (see Appendix A), the FDA supports the FD structure illustrated in Figure 6.2. The user must work within this structure, and cannot add any additional sections.

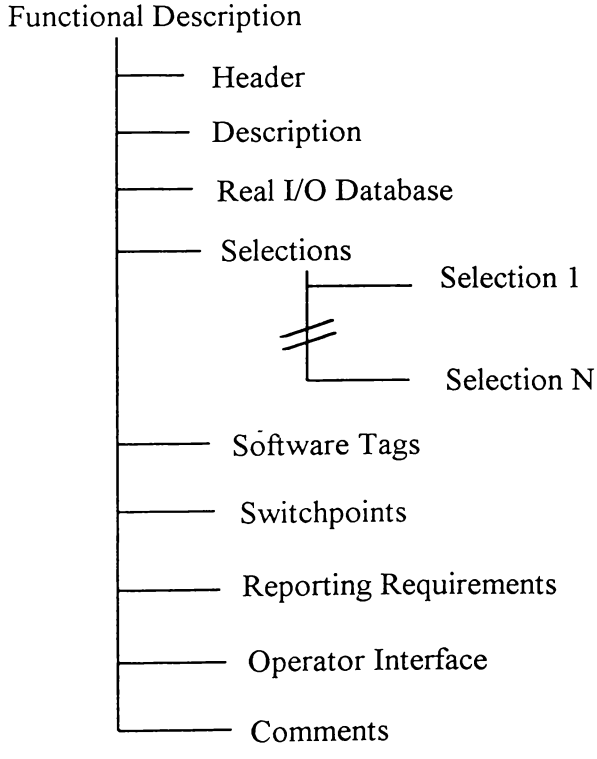


Figure 6.2. Standardised FD Structure Supported by the FDA.

The software tags and switchpoints sections are automatically generated and the real I/O data base can be automatically generated or manually created. Automatically

generated portions of the FD are created from information extracted from the batch control logic in the steps. Other sections are completed using a fill-in forms interface.

For the implementation of the prototype, effort concentrated on the implementing the selections section which contains the batch control logic. Forms, implemented using Window dialogue boxes, were developed for each section of the representative structure (from Figure 4.5). The hierarchical structure of the interface is shown in Figure 6.3.

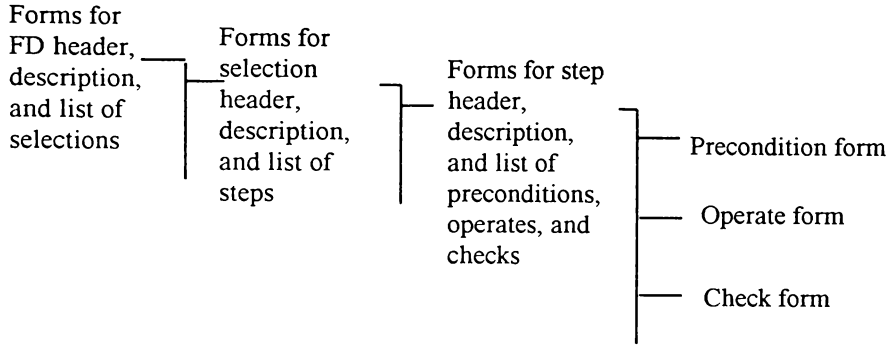


Figure 6.3. Structure of the Forms-Based Interface.

The user is restricted to this fixed hierarchy. Within each level of the hierarchical structure the user can create any number of FDs, selections, steps, etc.

The general syntax of the control commands is fixed. Operates generally take the form of <command> <device name>, such as Start P201, and preconditions take the form of <device name> <state>, such as P201 Running. Some "device names" that refer to the invariant portions of the structure, such as steps, selections, and the operator, are embedded into the CASE tool as are commands and states that refer to these names.

Another fixed portion of the specification concerns the naming of plant devices. Names must take the format <prefix><number>, where the <prefix> is an alphabetic tag that indicates the type of the device. For example P201 might indicate a pump and VP101 might indicate a valve.

The following summarises the fixed portions of the abstraction.

- The sections in the FD and their purpose.
- Hierarchical structure of the batch control selections.
- Structure of steps using preconditions, operates, and checks.
- General syntax of the batch command language.
- Naming conventions for manufacturing devices.

6.2.2 Behavioural Model -- the Variable Abstraction

Any number of FDs can be created, named, and stored in the FD repository. Within a FD, the user may create any number of steps, which are named and can be saved in the repository. Finally, the user specifies the batch commands within the preconditions, operates, and checks of each step.

Additional specialisation is provided by populating the manufacturing dictionaries and rules, shown in the CASE tool architecture of Figure 6.1. There are four dictionary/rule bases:

- Terminology dictionary and rules. This dictionary and rule base permits specification of the batch command language terminology to be used in the preconditions, operates and checks. It also specifies simple rules, in the form of state tables, that define the state changes that result from application of batch commands. For example, if it was desired that the commands Start and Stop be used with a pump, and that the corresponding operational states would be named Running and Not Running, these terms and their relationships would be entered into the terminology dictionary.
- Plant item type dictionary. This dictionary allows definition of the prefix codes for types of manufacturing devices and includes consistency-checking rules to be used by the CASE tool for those classes. Examples of rules include the command terminology permitted for each type of device, the number of inputs and output signals utilised by the device type, and whether the device is digital or analogue in

nature. An example is: VP is a prefix for a two-position valve with two feedback signals. Valid driving commands are Energ and Deenerg.

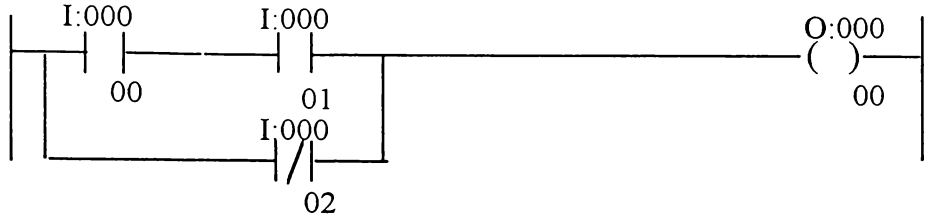
- Plant I/O dictionary. This dictionary permits specification of individual manufacturing plant devices. One could call entries in this dictionary instantiations of the types defined in the item type rule base. This dictionary is a repository of available devices and also contains a mapping between each device's I/O signals, associated batch commands, and corresponding PLC addresses. An example is: For valve VP101 the driving command Energ maps to PLC address O:110/00 and the states Energised and Deenergised map to PLC addresses I:060/00 and I:050/00.
- RLL Code Templates. This database contains RLL op-codes and code patterns that are used by the code generator to map the batch control specification in the FD to RLL code for the PLC. An example is: For valve VP101, the driving command Energ maps to RLL op-code OTE (output enable), and the states Energised and Deenergised map to the op-codes XIC and XIO. RLL code patterns are used when preconditions, operates, and checks are translated to RLL. The code patterns are filled in with the op-codes, and may also contain other code as appropriate.

Each individual rule base has an associated forms-based editor and repository. All of the editor programs were implemented using Visual C++ to operate under MS Windows.

6.2.3 Abstraction Realisation

The code generator produces RLL code for the Allen-Bradley PLC5. Since a direct interface to the PLC is proprietary, the ASCII import/export format for the code is produced.

Figure 6.4 shows an RLL rung and its corresponding ASCII export equivalent.




```
SOR BST
XIC I:000/00  XIC I:000/01  NXB
XIO I:000/02  BND
OTE O:000/00  EOR
```

Figure 6.4. Example of ASCII Import/Export Code for Allen-Bradley PLC.

Op-codes in Figure 6.4 are XIC (Examine Closed), XIO (Examine Off), and OTE (Output Energise). SOR and EOR indicate "Start Of Rung" and "End Of Rung", respectively, and BST, NXB, and BND indicate "Start of Branch", "Next Branch", and "Branch End". PLC data table addresses are coded as I:000/00, I:000/01, I:000/02, and O:000/00. I indicates the input image file and O indicates the output image, 000 indicates a word address, and /00, /01, and /02 indicate particular bits in a word.

For the prototype, it was decided to implement partial code generation to test the feasibility (or the lack of) and potential of automatic code generation in the domain. Thus, the scope of the code generator has been limited, as will be presented later in this chapter.

6.3 Specification and Manufacturing Rules

This section describes the implementation of the dictionaries and rules that allow specialisation of the command language and manufacturing information. In terms of the CASE tool architecture, the bold portions of Figure 6.5 are addressed in this section.

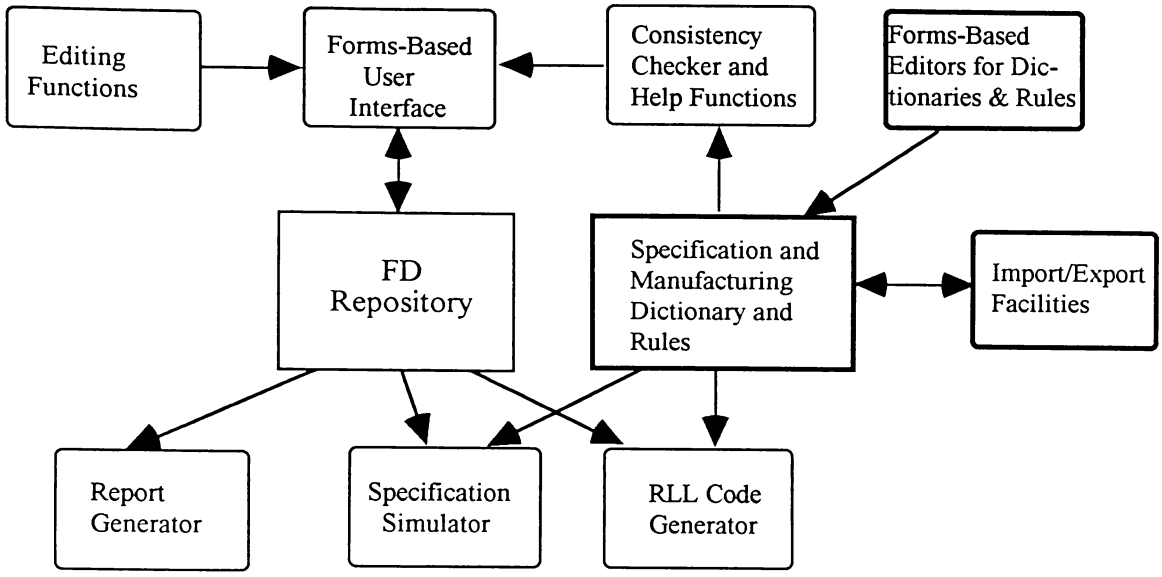


Figure 6.5. Specification and Manufacturing Dictionaries and Rules.

Terminology Dictionary and Rule Base

The Terminology dictionary and rule base is used to define the keyword operates and states that can be used in a functional description. For example, one FD writer may prefer to use the operates energise and deenergise when referring to valves but another may prefer open and close. Similarly, one engineer's preference may be to use the states energised and deenergised to correspond with the operates energised and deenergised, whilst another may prefer opened and closed. An example is shown in Table 6.1.

Operates	States	
	Energised	Deenergised
Energise	Energised	Energised
Deenergise	Deenergised	Deenergised

Table 6.1. Terminology Rule Base Entry

This example shows that applying the Energise operate to an item that is currently in the Deenergised stated will move that item into the Energised state. Conversely, application of the Deenergise operate will move an Energised item into the Deenergised state. These rules are utilised by the FDA to determine which states are permitted in preconditions and checks for each plant item type. They are also utilised for the model-based analysis feature of the FDA.

Operates that refer to analogue devices usually have more than two states. For these operates, only the operate terms are entered in the rule base and those terms are marked as not being "stateful". In reality, it means that they have multiple states and such states are defined in the plant I/O rule base which is described below.

The terminology rule base was implemented as an editor and corresponding data base. Figure 6.6 shows the top level dialogue box after opening a terminology rule base, and Figure 6.7 shows the dialogue for adding or editing an operate.

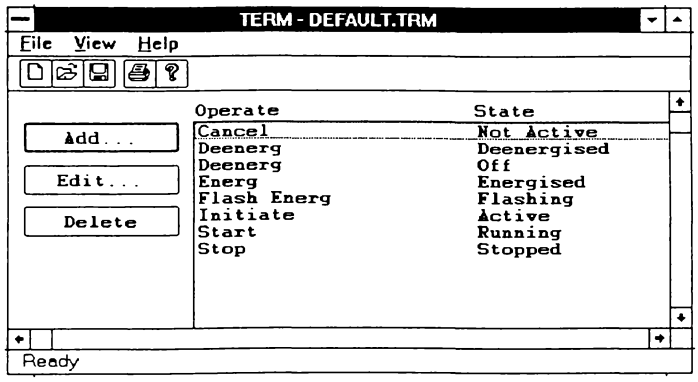


Figure 6.6. Top Level Terminology Rule Base Dialogue Box.

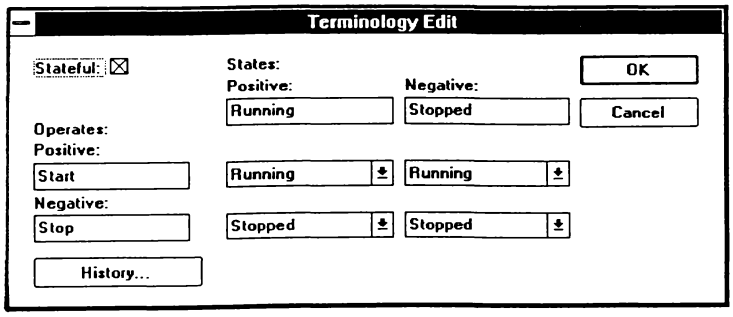


Figure 6.7. Terminology Rule-Base Edit Dialogue.

Plant Item Dictionary

The Plant Item Type dictionary and rule base allows the definition of data types that refer to types of devices that exist in the factory. Examples of plant items are VP, IN, and P as in VP101 (valve 101), IN401 (indicator lamp 401), and P201 (pump 201). The initial prefix code (eg. VP, IN, P) indicates a type; the numeric value indicates an instance of a type. Examples are shown in Table 6.2 below.

Prefix	Description	Type	Inputs	Outputs	Operates and(States)
VP	Two position valve with feedback	Digital	2	1	Open, Close, (Energise, Deenergise)
P	Pump	Digital	1	1	Start, Stop (Running, Not Running)
SW	Switch	Digital	1	0	(Energised)
IN	Remote Indication Lamp	Digital	0	1	Energise
LT	Level Transmitter	Analogue	1	0	(value)
FIC	Flow Controller	Analogue	1	1	Enable Auto, Enable Manual (Manual, Auto)

Table 6.2. Plant Item Type Information.

This dictionary permits the FDA to provide help and automatically supply some descriptive information. The FDA will provide a list of operates, states, and associated terminology for each particular type of plant item. For example, the FDA will list the operates open and close and the states energise and deenergise to be used with plant item VP101 due to its type (prefix VP).

The top level dialogue from the editor for the plant item rule base is shown in Figure 6.8.

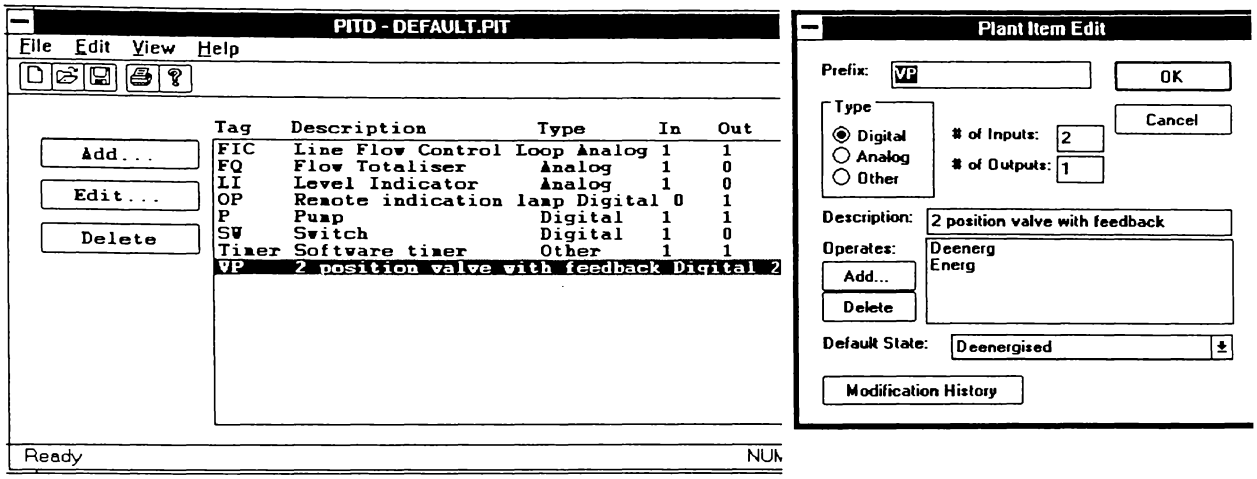


Figure 6.8. Top Level Plant Item Type Dialogues.

Plant I/O Dictionary Rule Base

The Plant I/O dictionary and rule base allows the association of individual plant items and their operates and states with specific real I/O addresses in the PLC. Examples of the information that can be stored in this dictionary/rule base are shown in Table 6.3.

Plant Item	Description	Equip Tags	Operate(s) or State(s)	PLC Addresses
VP101	Silo S1 CIP Supply	V101 V101 V101D V101E	Energise, Open Deenergise, Close Deenergiseed,Closed Energised,Open	O:110/00 O:110/00 I:060/00 I:050/00
P201	Milk Transfer Pump	P201 P201 P201R	Start Stop Running	O:100/00 O:100/00 I:040/00
SW301	Silo CIP Gateswitch	SW301	Energised	I:070/00

Table 6.3. Example of Plant I/O Information.

This rule base is utilised by the FDA to give the process engineer "pick lists" of plant item names that are available for use in the control specification. It is utilised by the code generator in order to translate preconditions, operates, and fault checks into corresponding RLL code that must reference PLC data table addresses.

Examples of dialogues for the Plant I/O editor are shown in Figures 6.9 and 6.10.

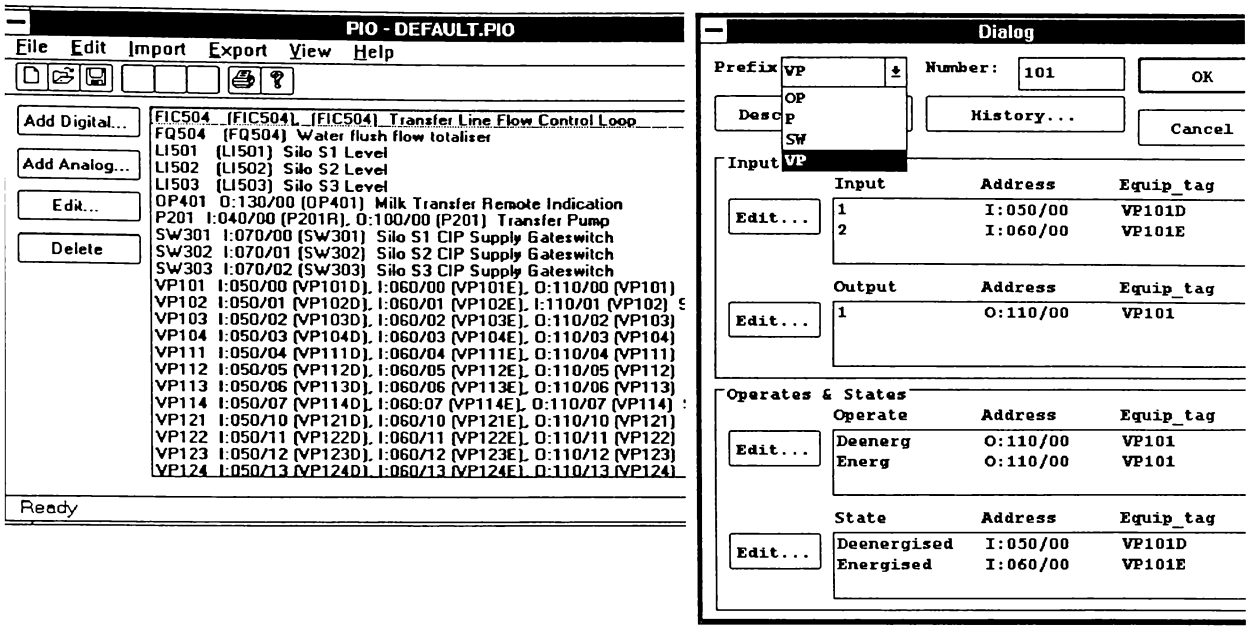


Figure 6.9. Top Level Plant I/O Edit Dialogues and Dialogue for Editing a Digital Item.

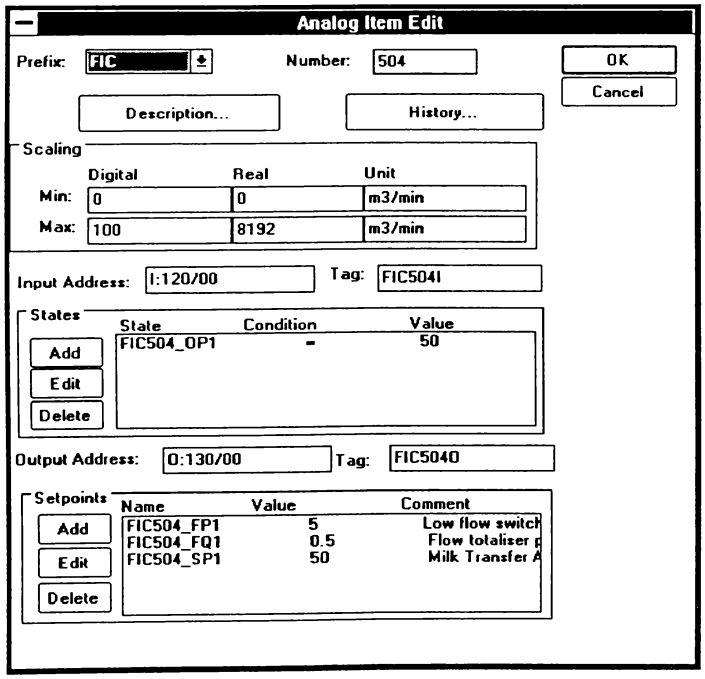


Figure 6.10. Dialogue for Editing an Analogue Type.

Although it was not implemented for the prototype, import and export of PLC I/O schedules would be supported in this dictionary. This would be used to interface the CASE tool with an external list of PLC address for plant items that might be supplied by the electrical contractor.

RLL Op-Codes and Templates

As was described in Section 6.3.3, the prototype methodology companion generates RLL code for Allen-Bradley PLC5 machines using Allen-Bradley's ASCII import/export representation. The code generator works by translating each batch control command into RLL through the use of code templates (or patterns) which contain the PLC instruction corresponding to the command. Also, overall structuring templates are used for preconditions, operates, and checks. Ultimately, these templates will be defined and maintained by the user to support customisation and extension of the code generator.

The following types of RLL templates are defined:

- A generic template for preconditions.
- A generic template for fault checks.
- Templates for each plant item type/condition combination to be used to complete the generic precondition and fault check templates.
- Templates for each plant item type/operate combination to be used for generation of code in the driving command output files.
- Templates for certain specific steps, such as step 0 that initialises internal variables, and the cancel step, that also does some special work.

Templates are composed of literal text and expansion tokens. Literal text is copied as it is encountered by the code generator. Examples are SOR, BST OTE, ST, NXB, etc. Expansion tokens are replaced (expanded) by the code generator into names of PLC addresses or into RLL code using other templates. Examples of tokens are asterisk (*) and any text enclosed in tildes (~). Table 6.4 describes the function of each expansion token.

Token	Expanded To
*	The RLL template for a precondition, operate, or check Example: "Step 2 Active" expands to XIC P101ST2
~item~	The name of a plant item. Example: VP101
~selection~	The name of the current selection. Example: P101
~stepno~	The current step number. Example: ST3
~statusmsg~	The message number for the current step name. Example: 8361 for "Start Transfer"
~faultmsg~	The message number for the current fault check. Example: 4000 for "Pump 201 Not Running"

Table 6.4. Examples of RLL Template Expansion Tokens.

Figure 6.11 shows generic templates for the step preconditions, in 6.11a, and fault checks, in 6.11b. (See Figure 4.13 for an example of actual RLL code.)

(a)
```
SOR *
  BST OTE ~selection~ST~stepno~ NXB
  MOV ~statusmsg~ ~selection~PSS NXB
  MOV ~stepno~ ~selection~PST BND
EOR
```

(b)
```
SOR
  XIC ~selection~ST~stepno~ * MOV ~faultmsg~ ~selection~PFT
EOR
```

Figure 6.11. Generic RLL Template for (a) Preconditions and (b) Fault Checks.

In Figures 6.11, the asterisk (*) is expanded into the RLL opcode and software tag corresponding to the condition to be tested by using another template (to be shown in Figure 6.13). Notice that the templates allow encoding RLL code for the setting of software flags, such as the step active bit, and storing of status and fault messages. For example, using the template in 6.11a, the precondition (for Step 3) "Step 2 Active .And. No Faults Step 2 Checks" would expand to the RLL shown in Figure 6.12.

```
SOR XIC P101ST2 XIC P101PFT/15
 BST OTE P101ST3 NXB
 MOV 8316 P101PSS NXB
 MOV 3 P101PST BND
EOR
```

Figure 6.12. Expanded RLL Code Using Template in Figure 6.11a.

The generation of RLL code for testing the state of steps and plant items (eg. "Step 2 Active" or "P102 Running") and operates (eg. "Start P102") requires additional RLL templates as shown in Figure 6.13. One of these templates is required for each plant item type/condition and for each plant item type/operate combination. Templates for conditions are shown in Figure 6.13a and for operates in Figure 6.13b.

(a)

Step	Active	XIC ~selection~ST~step~
Step	No Faults	XIC ~selection~PFT/15
P	Running	XIC ~item~R
P	Not Running	XIO ~item~R
VP	Energised	XIC ~item~E
VP	Deenergised	XIC ~item~D

(b)

```
P Start          SOR FON~item~ BST
                       *
                 BND FOF~item~ OTE ~item~ EOR

VP Energ         SOR FON~item~ BST
                       *
                 BND FOF~item~ OTE ~item~ EOR
```

Figure 6.13. RLL Templates for Item Types and (a) Conditions and (b) Operate Combinations.

The templates in Figure 6.13a are used to complete the precondition and check templates shown in Figure 6.11. For example, the precondition "Step 2 Active" expands into "XIC P101ST2" using the first template in 6.13a, and the check "P102 Running" would translate to "XIC P101R" using the third template shown in Figure 6.13a. Figure 6.13b shows instances of templates to be used to generate the RLL output files corresponding to the operates (see Figure 4.14). Notice that the force-on and force-off flags are incorporated into the templates, as well as the operate op code (eg. OTE) to be used to actuate the device. In these templates the asterisk is expanded

to test the step active flag (using the "Step Active" pattern) for each step that applies an operate to that device.

The logic of how to process each token is embedded into the fixed part of the code generator. The user cannot add new tokens to the code generator. Also, algorithms concerning the processing of compound conditions (.And., .Or., etc.) are embedded into the code generator.

6.4 Functional Description Assistant

The Functional Description Assistant (FDA) is the CASE tool designed to support development and analysis of the control system specification, the Functional Description (FD). It is to be utilised primarily by the process engineer and replaces generic office automation tools (word processor) in the old system.

In the following sections, the menu-driven, forms-based user interface will be reviewed including consistency checking and help functions, followed by the editing functions, the report generator, the simulator, and the code generator.

6.4.1 User Interface, Consistency Checking, and Help Functions

This section discusses the user interface to the FDA and reviews domain-specific helps and consistency checking provided by the tool. These features are represented by the bold modules in Figure 6.14.

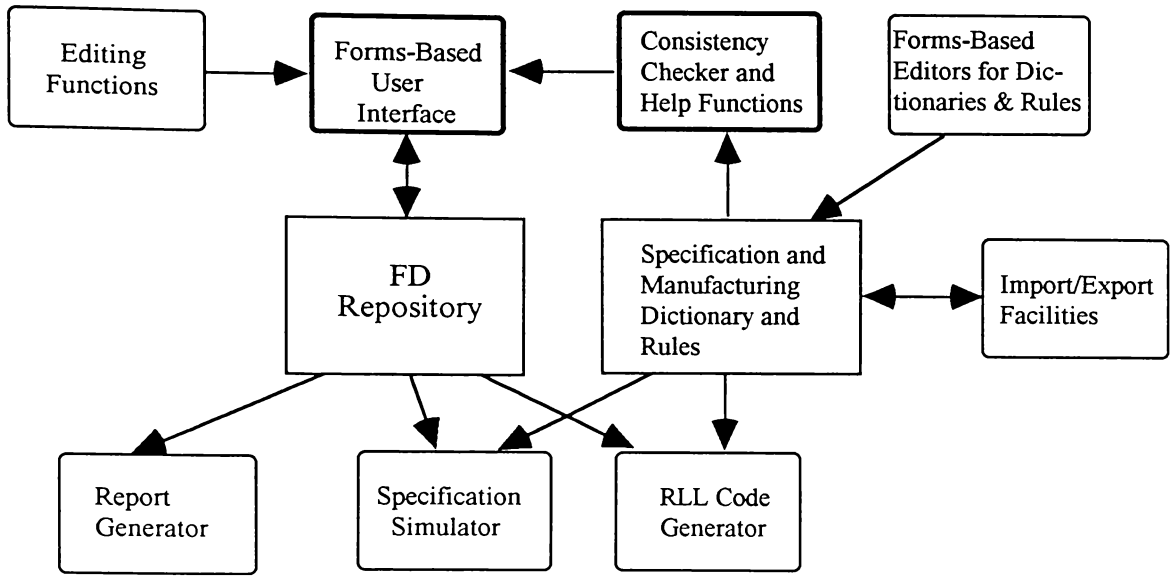


Figure 6.14. User Interface Functions.

The initial screen of the FDA is shown in Figure 6.15 after the existing FD named SAMPLE has been opened. Using the File menu, the user can choose to create a new FD (using New) or Open an existing FD (using Open) using the typical Windows conventions. Combining Open with Save As provides one way to reuse existing FDs -- the user can open an existing FD, modify it, then save it as a new FD using Save As. Clicking on the FD choice on the FDA's menu bar shows the menu corresponding to the FD structure. (See, the right side of Figure 6.15.) The menu item "Schematic" is intended to provide support for representation of the Process and Instrumentation Diagram (P&ID), but this was not implemented in this version of the software. The other items on the FD menu are described later in this and the remaining sections of this chapter.

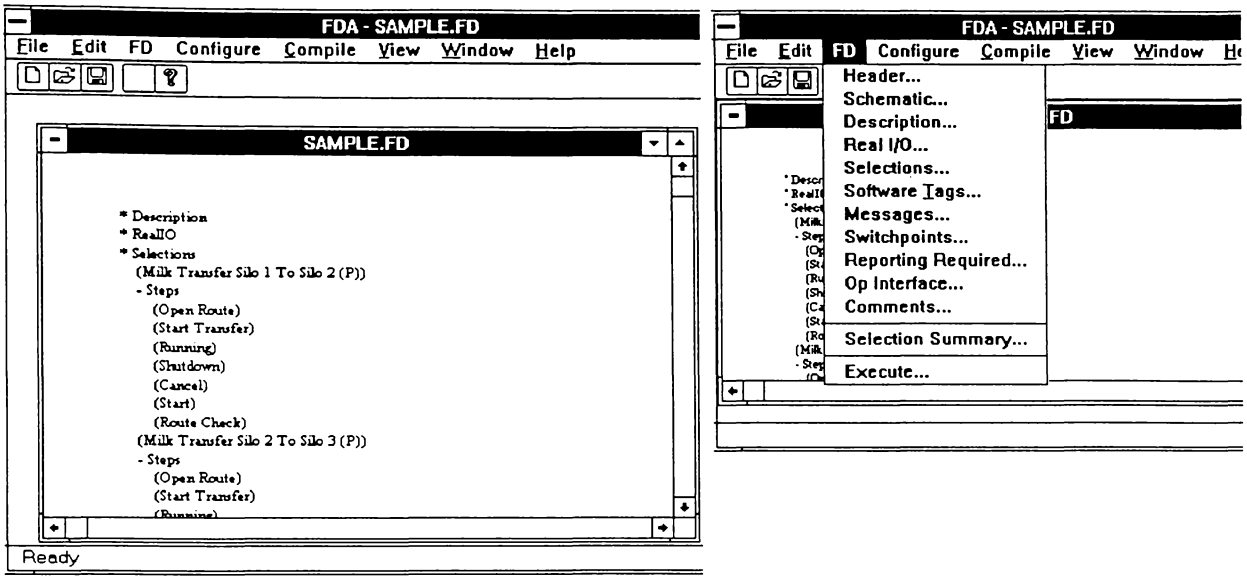


Figure 6.15. Initial Screen After Opening an Existing FD (left), and After Opening the FD menu (right).

The header specifies the name of the FD, its number, and provides for a comment field. The description is a text box in which the user enters a narrative description of the process. Clicking on these menu items brings up the dialogue boxes shown in Figures 6.16 and 6.17.

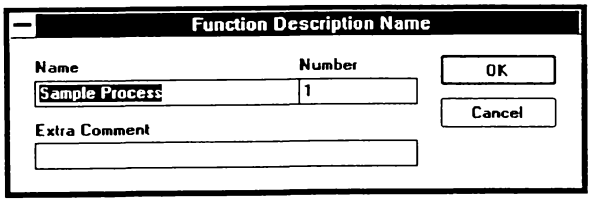


Figure 6.16. Header Dialogue Box.

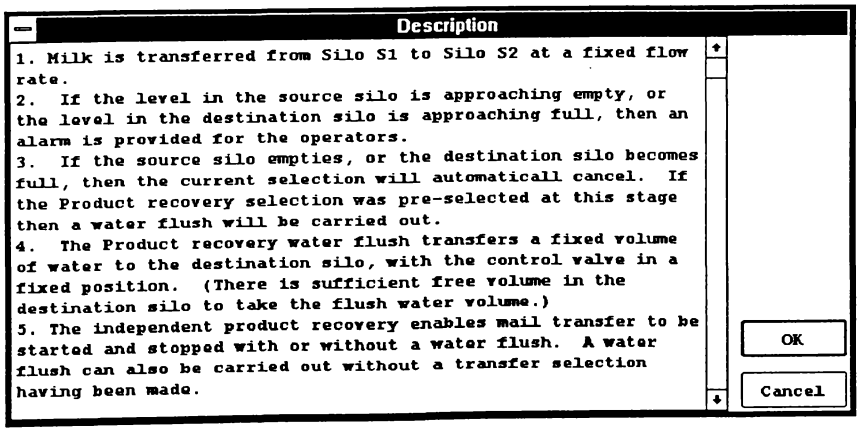


Figure 6.17. Description Dialogue Box.

The Real I/O database is a listing of all plant items (and an associated description) that are (or will be) referenced in the selections. The Automation Database can be constructed prior to entering selections, or automatically by the FDA as selections are entered. For example, as the user is entering preconditions, operates, or checks, if a plant item is referenced that is not in the Real I/O Database, the FDA will automatically enter it. Descriptions are automatically extracted from the Plant I/O dictionary and item type attributes are automatically extracted from the Item Type dictionary. The latter can be viewed by clicking on the Attributes button in the Real I/O dialogue that is illustrated in Figure 6.18.

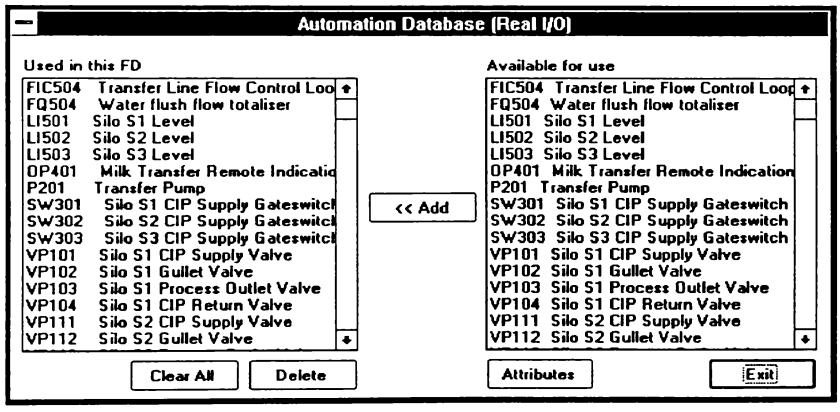


Figure 6.18. Real I/O Dialogue Box.

The heart of a FD are the selections, which specify the logic of the control specification. The structure of the selections portion of the specification is fixed using the representative hierarchical structure. As was illustrated in Figure 6.3, fill-in forms, implemented as dialogue boxes, are provided for each level of the hierarchy. The interface is illustrated below.

FDs consist of one or more selections. The dialogue in Figure 6.19, which shows this list, is displayed when the "Selections..." is chosen from the FD menu (see Figure 6.15).

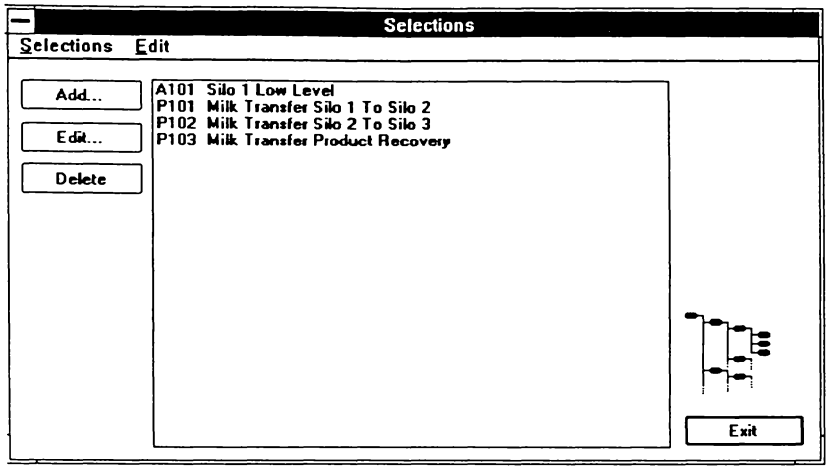


Figure 6.19. Selection Dialogue Box.

The selection dialogue box (Figure 6.19) is used to add, edit, or delete selections from the FD. For example, pressing the Add (or Edit, for an existing selection) in this dialogue box brings up the step dialogue box which is illustrated in Figure 6.20.

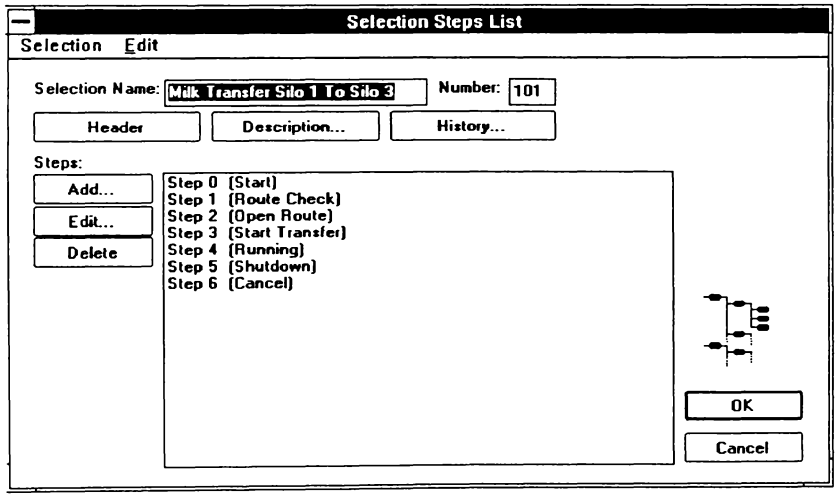


Figure 6.20. Step List Dialogue Box.

Notice the bit map icon in the lower right of the dialogue box. This is a visual cue of the current position in the selection, step, precondition, operate, check hierarchy.

Each selection has a name, number, type, header, description, modification history, and one or more steps, which are entered using the corresponding fields and buttons in Figure 6.20. The header dialogue box (see Figure 6.21), which pops up when the Header button is pressed, is used to capture a bit of information used for code generation known as preselection criteria (the selections that may start (preselect) this one) and also the type of selection, which is either process (P), alarm (A), clean in place

(C), or background (B). Currently this information is used as a convention, but it is anticipated that specific information about each type of selection may be embedded into the FDA (or the code generator) in the future.

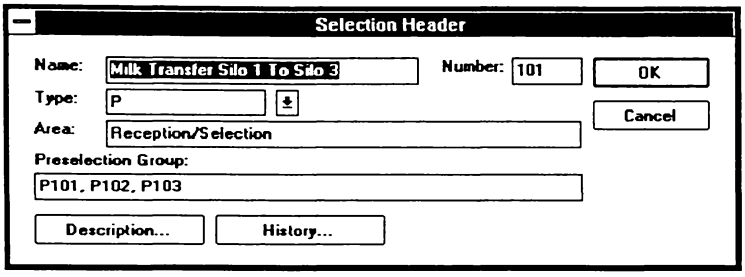


Figure 6.21. Header Dialogue Box

Pressing the description button pops up a dialogue box in which the user can enter a narrative description of the selection. (An example of this dialogue has been omitted.)

Clicking the History button pops up a dialogue (see Figure 6.22) which is used for building an audit for change control. Separate history records are provided throughout the selection and step hierarchy. Thus, each time a selection or step is modified, the user can make an entry in the history dialogue (the FDA automatically supplies the date) to record information pertinent to the change. The history entries are also displayed in the selection and step reports, that are described later in this chapter.

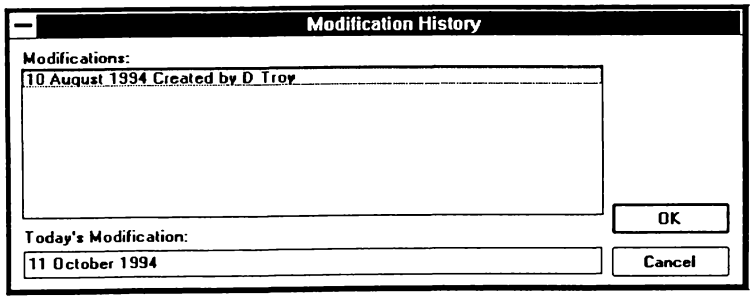


Figure 6.22. Modification History Dialogue Box.

Referring to Figure 6.20, clicking the Add button, or selecting an existing step and pressing Edit (or double clicking on the selection) will bring up the step dialogue box showing the preconditions, operates, and checks for the step. This is illustrated in Figure 6.23.

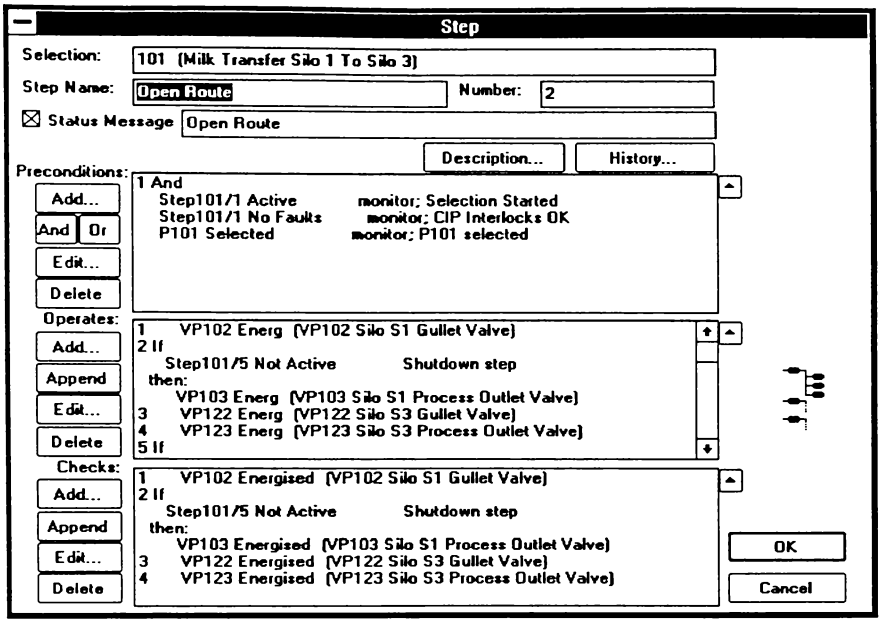


Figure 6.23. Dialogue Showing Details of a Step.

The logic for the preconditions, operates, and checks are created by filling in forms using dialogue boxes in each of these three sections.

In the precondition dialogue is shown in Figure 6.24, notice the down-arrow buttons near the Item and State boxes. Clicking the arrow-button next to the Item field produces a drop-down list giving the user a choice of any item in the Real I/O Database. Once the item is entered (or selected from the list), the FDA uses the item's type to produce a list of valid states by consulting the terminology dictionary, which can be viewed by clicking the arrow-button next to the State field. These drop-down lists are examples of domain-specific helps provided by the FDA. The Event and Monitor radio buttons are used to specify whether the precondition is to be continuously monitored (monitor) or if it is a one-shot condition (event). A comment field is provided for descriptive purposes.

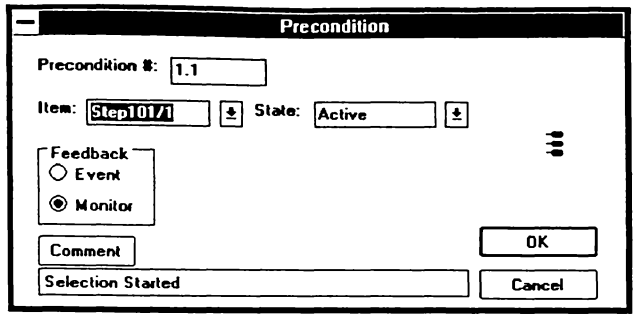


Figure 6.24. Precondition Dialogue Box.

The operates dialogue box is illustrated in Figure 6.25.

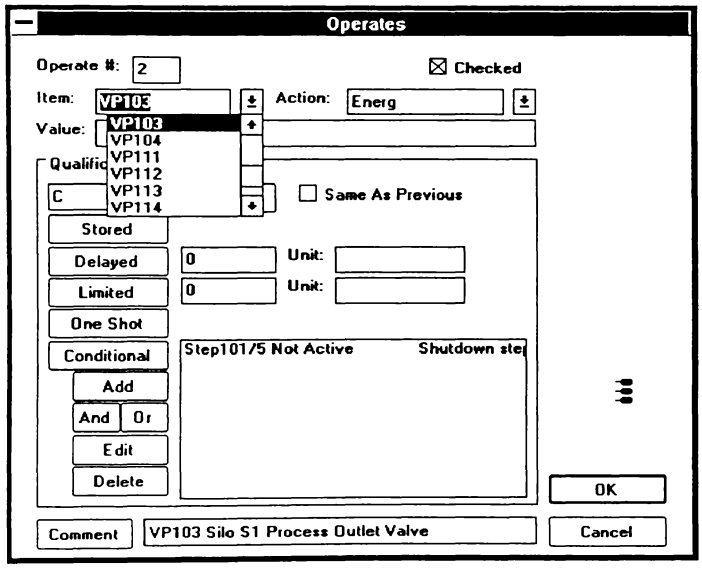


Figure 6.25. Operate Dialogue Box.

The operate number is automatically supplied by the FDA, but the user can enter any number desired. The Item field is a drop-down list, which, when the arrow button is clicked, displays a list of items available in the Plant I/O dictionary. Once an item is selected from the list (or entered), its valid operates are extracted from the Item Type dictionary and can be displayed in the drop-down list associated with the Action field. The Value field is used for messages and analogue type items so that the user can enter an appropriate value.

The Qualifications section of the dialogue was derived from ideas in Grafcet as described in (David, 1992). A stored operate is latched on and must be explicitly unlatched in a later operate. Operates can be either time delayed or time limited, thus making use of the PLC's timers. A one-shot operate is enabled for only one scan of the

PLC. Finally, operates may be conditional. These qualifications can be combined, and the ordering in which they are combined is important. The ordering is shown in the box next to the Delete button (partially obscured under the drop down list in Figure 6.25), and this button is used to remove qualifications. As an example of the importance of ordering, consider Delayed/Conditional versus Conditional/Delayed. In the former, the condition is applied after the specified delay, whereas in the latter the delay is applied only after the condition is satisfied.

A comment field is provided for descriptive purposes.

Clicking the Checked button near the top of the Operate dialogue will cause the FDA to automatically generate a fault check for that operate. Fault checks are described next.

The Fault Check dialogue is shown in Figure 6.26.

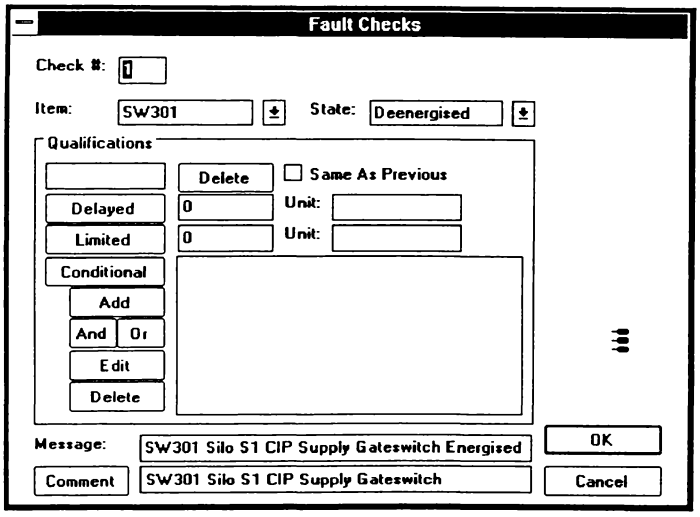


Figure 6.26. Fault Check Dialogue Box.

A default check number is automatically supplied by the FDA but the user may enter a number explicitly. To obtain help when entering the item and its expected state, the user can use the drop down lists or enter a value. Once the item is selected (or entered) the FDA will supply a list of appropriate states (based on the item's type). As with operates, checks can be qualified, time delayed, or time limited. The message field is used to capture a fault message that should be displayed on the Man-Machine Interface and is automatically generated by using the item's description and the opposite state

from the terminology rules. Finally, the comment field in Figure 6.26 is used solely for descriptive purposes.

6.4.2 Additional Editing Functions

Editing functions are meant to supply functionality similar to the word processor's find, replace, and copy/paste functions. However, in the FDA they are structured and can supply more succinct information. The bold module in Figure 6.27 shows this function's place in the CASE tool architecture.

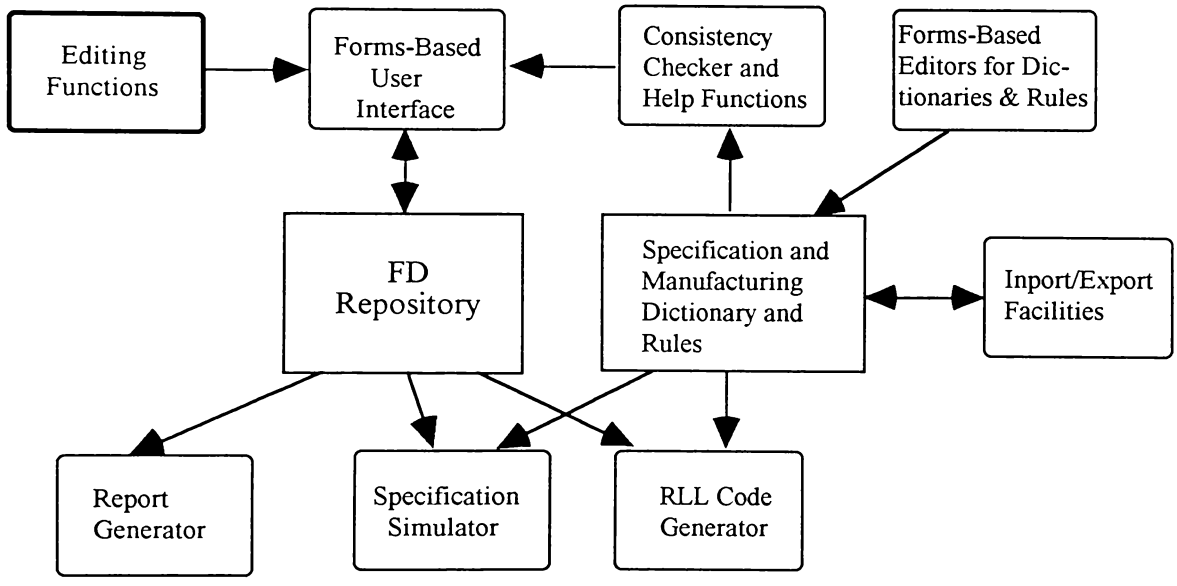


Figure 6.27. Editing using Copy, Paste, Find and Replace.

These functions are invoked using the Edit menu in either the Selections dialogue (see Figure 6.28) or in the Steps Dialogue (see Figure 6.20). As Figure 6.28 illustrates, the Copy/Paste function allows copying of entire selections, individual steps, or of items in the Real I/O list from one selection or step to another. Copy/Paste is much simpler than in the word processor, because the user needs only to click on the name of the selection or step to be copied, and the FDA will automatically copy the entire object. Pasting this into another FD requires renaming and renumbering of the selection, for which the FDA will prompt the user.

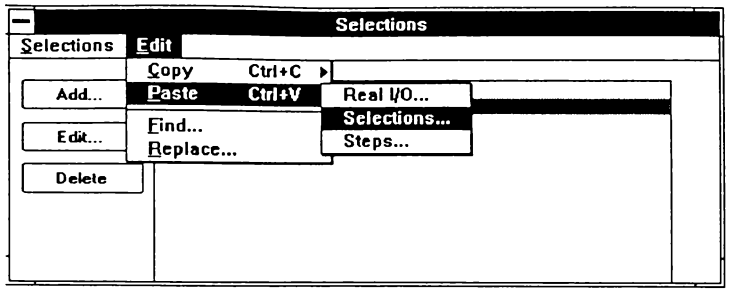


Figure 6.28. Copy/Paste for Selections.

The replace feature enables changing of plant item names, as illustrated in Figure 6.29. It can operate on selections or steps as well as on the Real I/O list, and it supports both global replacement and an interactive mode that requires the user to confirm each replacement.

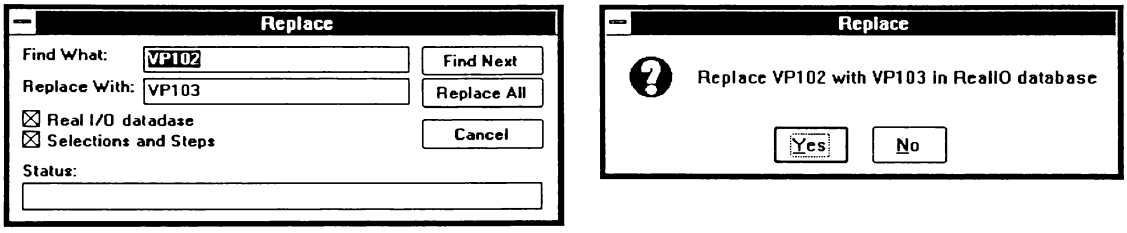


Figure 6.29. Replace Dialogues.

The find function is a design aid to help the user quickly locate where, in an FD, a plant item is referenced. As Figure 6.30 shows, Find informs the user of the step names, whether an item is used in a precondition, operate, or check, and the corresponding command along with the item's description which is extracted from the Real I/O database.

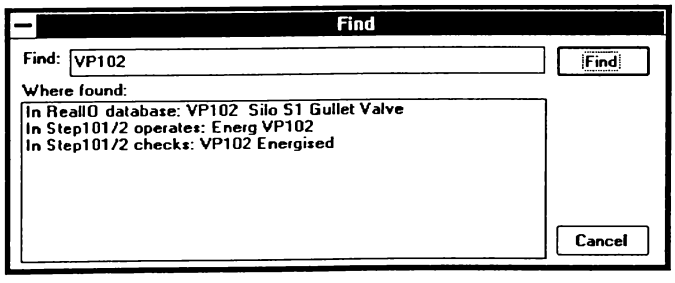


Figure 6.30. Find Dialogue.

Another design aid of the FDA are the reports produced by the FDA. These are the topic of the next section.

6.4.3 Report Generator: Design Analysis and Documentation

The FDA supports of analysis of the design specification in two ways: automatic generation of reports that support inspection of the design and model-based analysis of the specification. This section presents the report generation capabilities, represented by the bold module shown in the architecture of Figure 6.31.

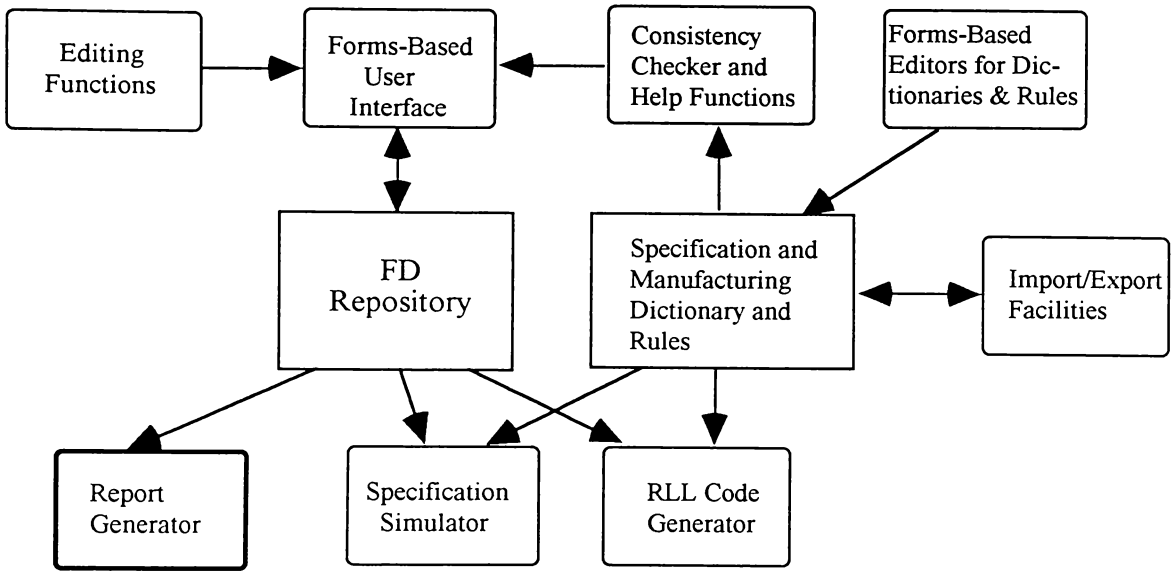


Figure 6.31. Design Analysis and Project Documentation Report Generator.

The reporting functions are found in the File/Report Select menu shown in Figure 6.32.

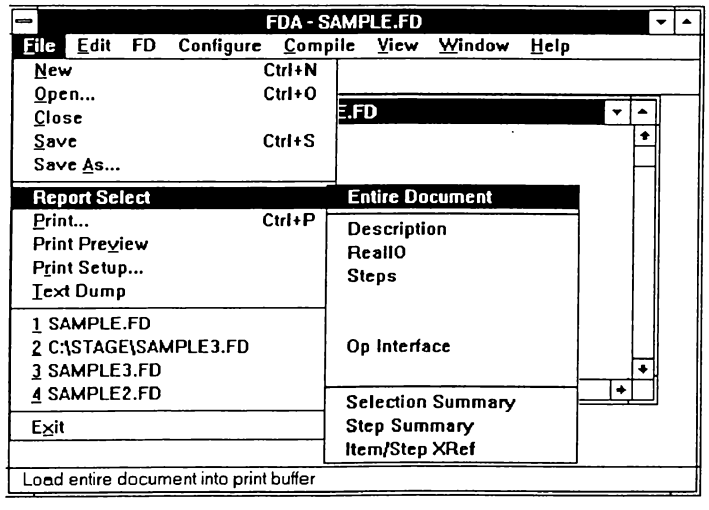


Figure 6.32. Report Selection Menu.

The FDA can be used to print the entire FD to produce the traditional complete FD document, as well as portions of the FD and other reports such summary or cross references. Available reports are:

- Messages. A list of all the messages (status, fault, and display) referenced in the FD.
- Software Tags. These are flags that the software engineer has traditionally included in the PLC code. An example is the "active bit" that is set for each active step.
- Selection Summary. A summary report listing the name, number, and descriptions of every selection and in a FD.
- Step Summary. A summary report listing the name, number, and descriptions of every step in a selection.
- Item/Step Cross Reference. A report showing each plant item and in which step the item has been referenced.

Examples of these reports are shown in Figures 6.33 through 6.36.

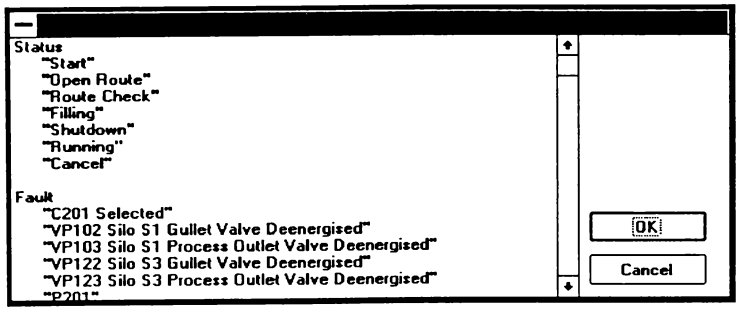


Figure 6.33. Messages Report.

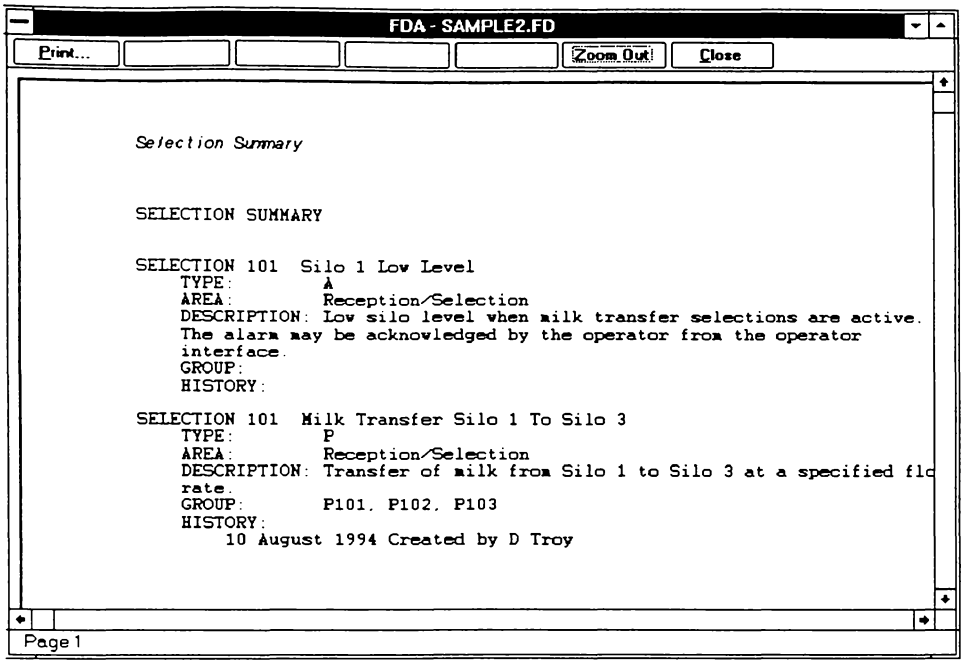


Figure 6.34. Selection Summary.

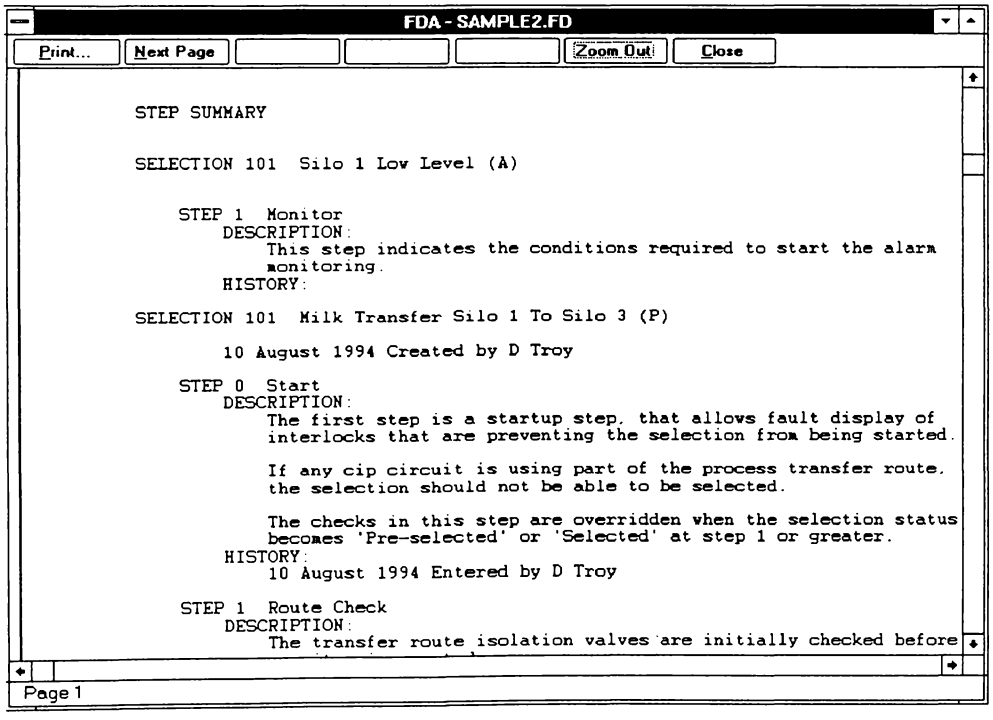


Figure 6.35. Step Summary.

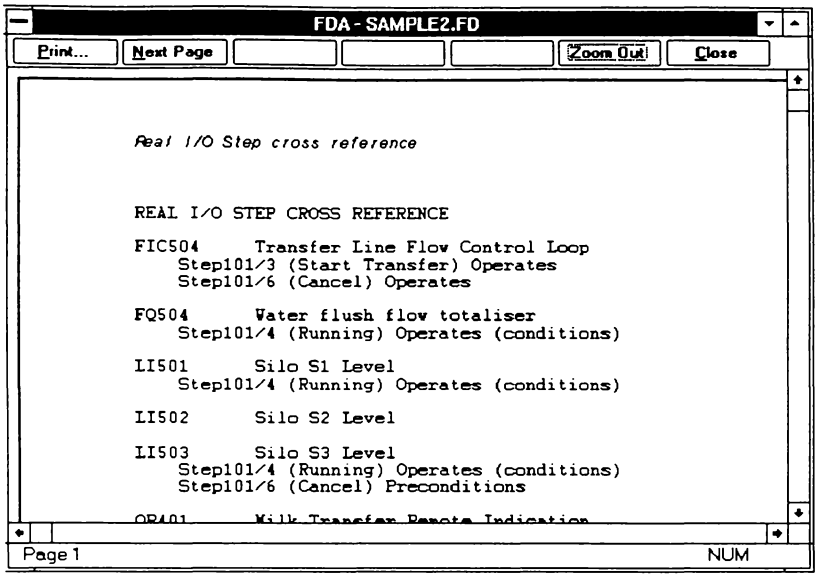


Figure 6.36. Item/Step Cross Reference.

These reports are intended to assist the process engineer in the following ways. They should increase productivity by replacing manual production with automation. Also, because of the ease with which they can be generated, the designer can produce intermediate versions of these reports as he/she is developing a FD to assist them with the design process. In the old system the designer may avoid production of intermediate versions of these reports because of the time required to do so.

6.4.4 Model-Based Analysis Using Simulation

The second kind of design analysis that the FDA supports is model-based analysis: the simulation (execution) of the FD control logic. This is the functionality of the bold module in Figure 6.37. The goal of this feature is to permit the designer to test the relationship of steps and their constituent preconditions, operates, and states by manually changing certain states and then letting the FDA simulate the execution of preconditions, operates and checks. The dialogues below illustrate the ideas.

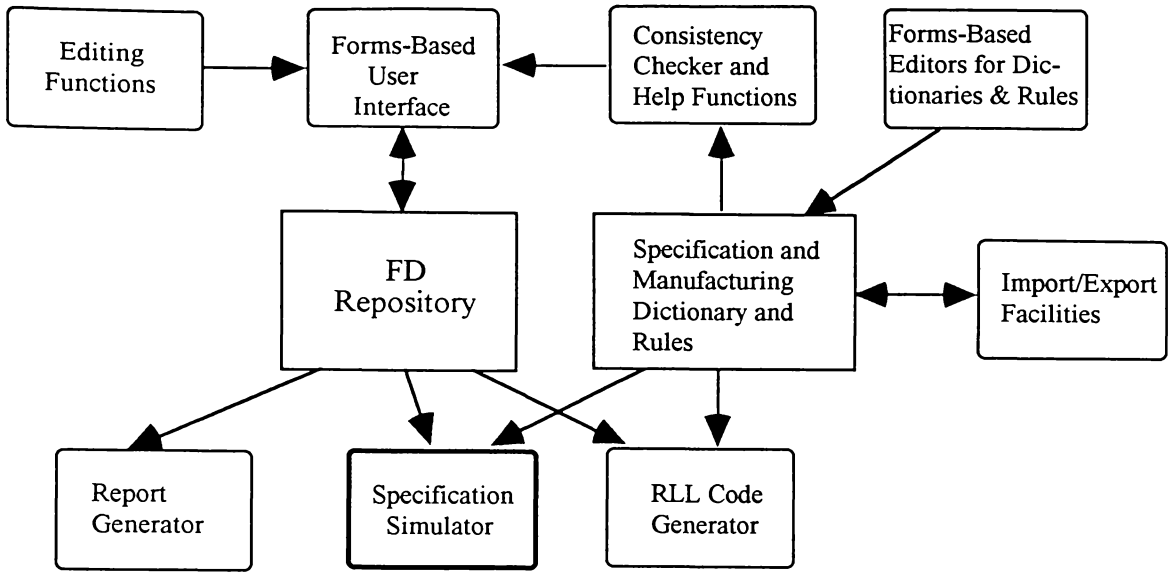


Figure 6.37. Simulation Module.

Clicking on the Execution menu pick under the main menu's FD menu item displays the selection execution dialogue shown in Figure 6.38. This dialogue displays the list of selections, allowing the designer to choose a selection with which to work and also to monitor the state of selections or "force" them into new states. This is shown in the Figure 6.38 where the selections are in their initial state of "not active."

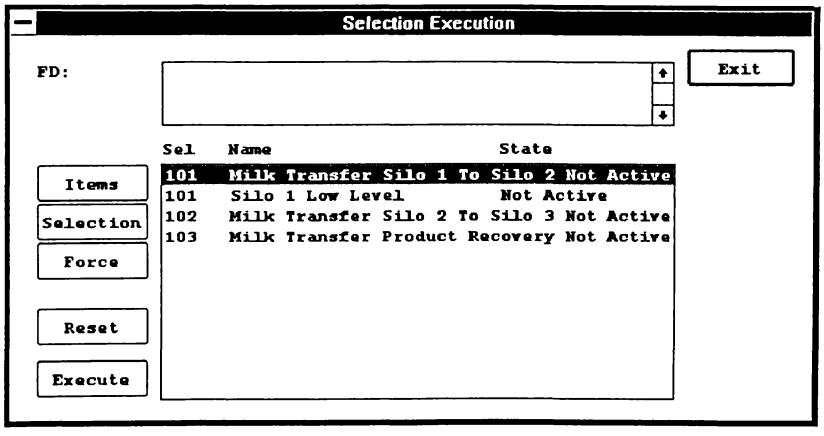


Figure 6.38. Selection States.

Highlighting a selection and pressing the "Selection" button allows the user to monitor the state of the constituent steps of the selection and also to force them into different states as is shown in Figures 6.39 and 6.40.

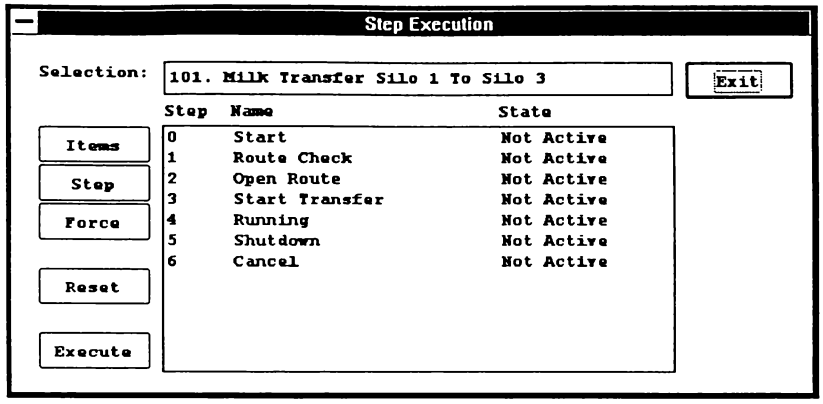


Figure 6.39. Examining Step States.

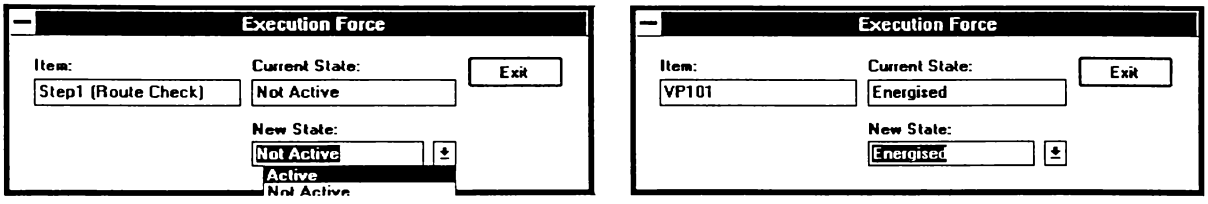


Figure 6.40. Forcing the State of Steps or Items.

Clicking on the Items button in Figures 6.38 or 6.39 displays a list of plant items used in the FD. This list, as illustrated in Figure 6.41, shows the current simulated state of each item and also includes a Force button to change an item's state, and a reset button to reset an item to its default state.

After forcing the state of steps or individual plant items in order to model an operational scenario, pressing the "Execute" button in Figures 6.38 or 6.39 will cause the FDA to simulate the step logic and update the states of the selections, steps, and plant items to reflect the logic in the preconditions, operates and checks. After execution, Figures 6.38 and 6.39 would be updated to reflect any state changes in selections and steps, respectively, and the updated item states can be examined by pressing the Items button in Figure 6.38 (or 6.39) to again view the state of each item (see Figure 6.41).

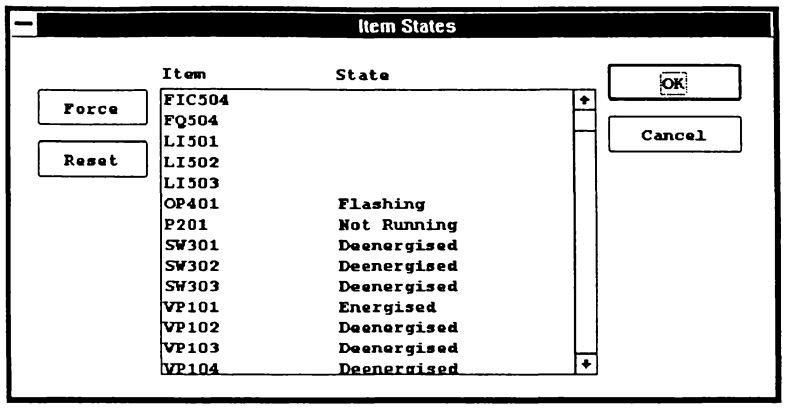


Figure 6.41. Updated State of Steps after Execution.

6.4.5 Automatic Code Generator

The code generator, represented by the bold module in Figure 6.42, transforms the FD into RLL code.

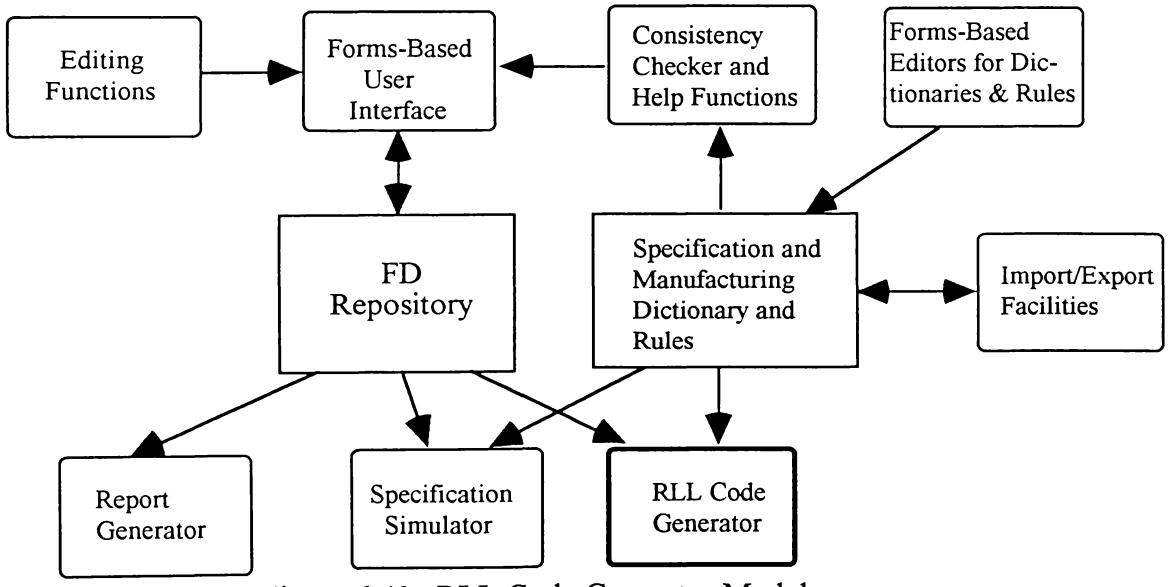


Figure 6.42. RLL Code Generator Module.

For the prototype, partial RLL code for the Allen-Bradley PLC5 was generated in ASCII import/export format, using the conventions and template approach that was described in Section 6.3.

Prior to generation of code, the user must invoke the configuration menu of the FDA in order to specify starting numbers for assignment to status, fault, and display messages, and also to give the starting PLC memory addresses for software tags. The configuration menu and associated dialogues are shown in Figures 6.43 through 6.45.

(The use of message numbers and software tags in RLL coding was introduced in Sections 4.6 and 6.3.)

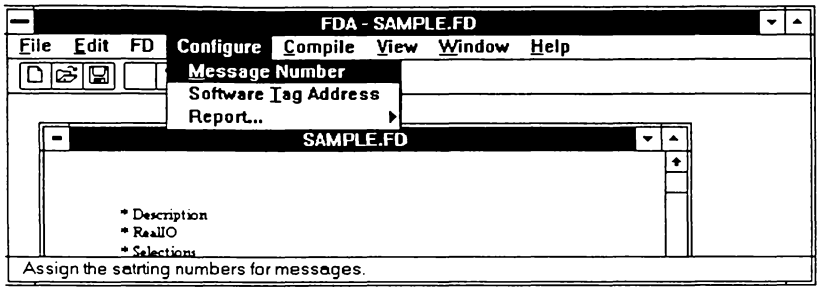


Figure 6.43. Configuration Menu for the Code Generator.

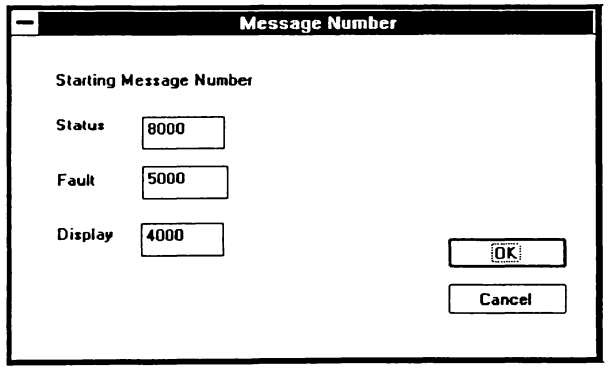


Figure 6.44. Configuring Message Numbers.

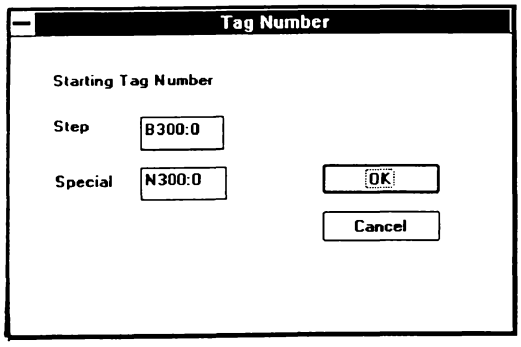


Figure 6.45. Configuring PLC Addresses for Software Tags.

In the examples above, numbers are assigned to status messages beginning with 8000, fault messages beginning with 5000, and display messages beginning with 4000. In Figure 6.45, the values B300:0 and N300:0 indicate the starting PLC memory address to be assigned to software variables (tags) that represent binary valued items (such as step active flags) and numeric valued items (such as the current status message), respectively. Following configuration, the code generator automatically

assigns numbers and addresses to all software tags and messages, and the results can be viewed using the Reports menu in the Configuration section. These reports are illustrated in Figures 6.46 and 6.47 below.

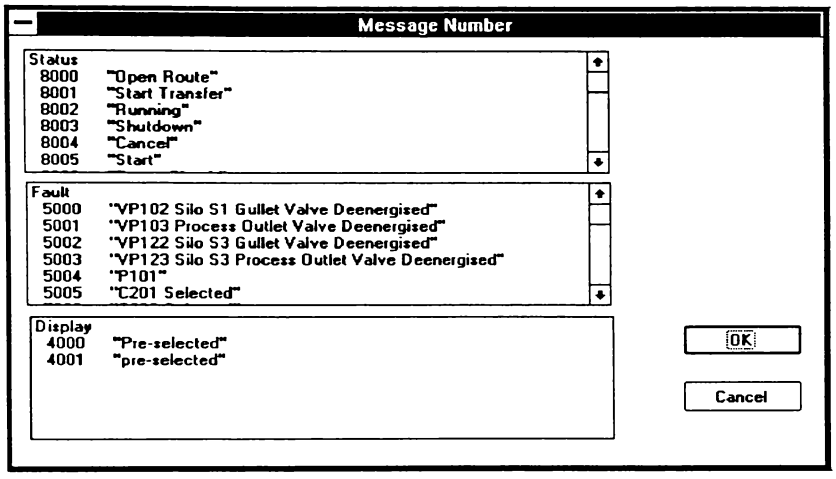


Figure 6.46. Message Numbers Automatically Assigned.

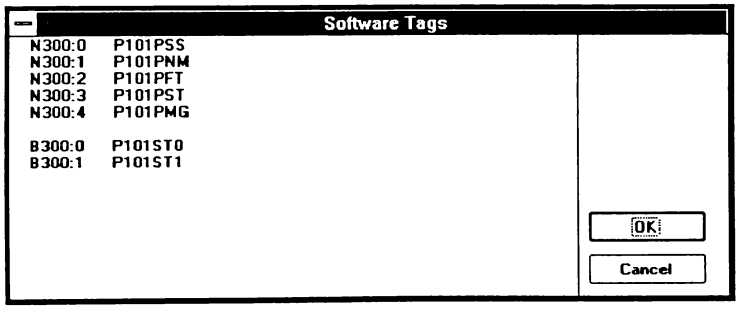


Figure 6.47. Software Tag Addresses Automatically Assigned.

Code generation is invoked via the Compile menu, which is shown in Figure 6.48.

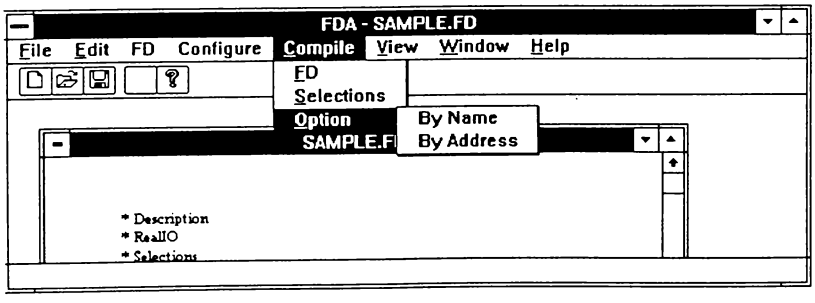


Figure 6.48. Compile Menu.

Options available for code generation include generation of code for either the entire FD or for an individual selection. Also, the code can be generated using symbolic names for tags (such as P101ST1) by using the "By Name" option, or the code can be

generated with PLC memory addresses (such as B300:1). The code generator retrieves the real I/O address of plant items from the Plant I/O dictionary, and combines the addresses with the op-code mapping's and RLL templates to produce the RLL code in ASCII import/export format.

As an example of the code generated, consider the FD fragment shown below in Figure 6.49.

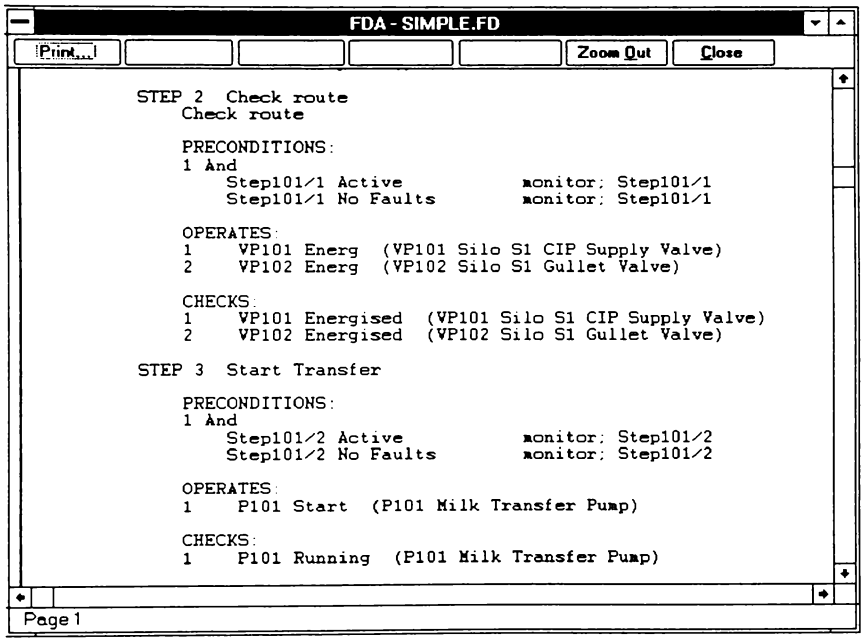


Figure 6.49. FD Fragment.

For the FD fragment shown in Figure 6.49 the code generator produces several ASCII files: SIMPLE.RLL which contains the code for the precondition and check logic, VP.RLL which contains the operates for VP items, and P.RLL which has the operates for the pump. The code for the preconditions, generated by name, is shown in Figure 6.50. Generation of code by address would replace symbolic names for software tags and variables with PLC addresses.

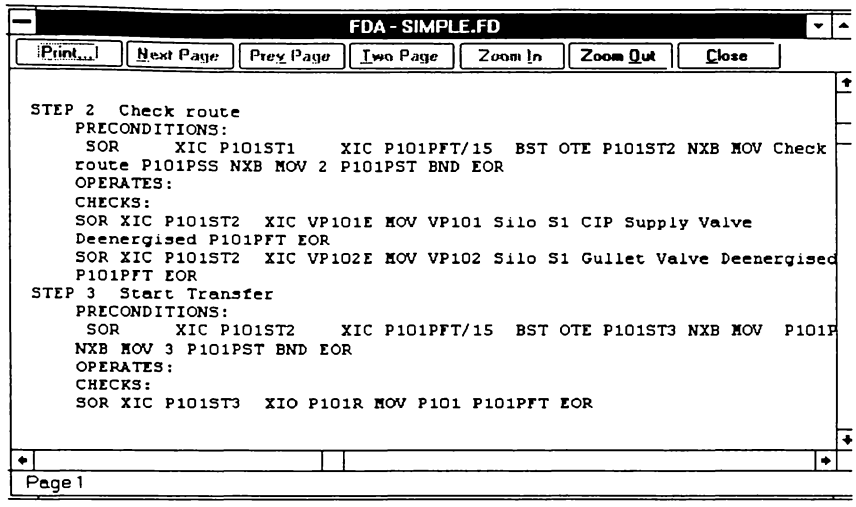


Figure 6.50. RLL Code for Preconditions and Checks.

The code in Figure 6.50 illustrates the testing and setting of the precondition bits, as well as status messages and step numbers. This follows to the coding conventions described in Section 4.6 and illustrated in Figure 4.13. Also in keeping with the coding conventions at Anchor, the operate code for particular devices is written in separate ASCII files for each device type. For example, the generated code for VP devices is shown in Figure 6.51, which follows the conventions described in Figure 4.14.

SOR BST XIC FONVP102 NXB XIC P101ST2 BND XIO FOFVP102 OTE VP102 EOR
SOR BST XIC FONVP103 NXB XIC P101ST2 XIO P101ST5 BND XIO FOFVP103 OTE VP103 EOR

Figure 6.51. Operates Code for VP Devices.

In Figure 6.51 notice that the force-on/force-off bits are generated as called for by Anchor's coding conventions. Similar code is generated for other devices, such as the pump.

6.5 Summary

The results of this chapter represent the second major cycle of the research: the implementation of the prototype methodology companion. It was developed using a series of design-code-demonstrate cycles between the author and engineers and managers at Anchor and also with a process engineer, which allowed the author to demonstrate design concepts and to collect feedback from the users concerning their

correctness and appropriateness. The process served to formalise the functional description language and specification approach and to incorporate feedback into the tool.

The prototype methodology companion is composed of a set of dictionary/rule bases that support customisation of the tool and the Functional Description Assistant that supports specification and analysis of functional descriptions and also incorporates a prototype RLL code generator to transform the FD to PLC code.

The next chapter describes the assessment of the CASE tool in order to evaluate the underlying framework of concepts and to investigate the potential influence of the CASE tool on the practice of developing manufacturing control code.

Chapter 7 Assessment

This chapter presents the assessment of the methodology companion, its underlying framework of ideas, and its potential influence on industrial practice. The assessment methodology is reviewed, and then the results are presented.

7.1 Approach

The assessment approach is to first define the research aims of the investigation in terms of propositions and expected outcomes, introduce the prototype CASE tool to users in industry, and then obtain their informed judgment concerning the propositions and expected outcomes. This information is collected through structured interviews with a variety of users at various positions in the industry. The interviews were recorded, transcribed, indexed, and categorised and were then used to assess the research propositions. Each of these steps is described in the following sections.

7.2 Expected Outcomes

Recall the elaborated research proposition: Computer aided software engineering (CASE) tools that are domain specific, incorporate model-based analysis and automatic code generation, incorporate an appropriate methodology, and standards will be effective in supporting the software development life cycle for manufacturing control software for programmable logic controllers (PLCs). Effectiveness of the CASE tools should be reflected by:

- Improvements in production: The CASE tools will contribute to the capacity of individuals to accomplish development tasks with fewer resources. Productivity of individuals will improve and the quality of their work will also improve.
- Improvements in coordination: The CASE tools will contribute to control and maintenance of the design specification. The CASE tools will improve efficiency and effectiveness of individuals by enhancing their capacity to exchange information with others for the purpose of affecting the concept, process, or product of the development life cycle.

- Improvements in organisational technology: In comparison with the generic tools, the CASE tools will be easier to learn and use. The enforcement of standards will enable portability of skills, knowledge, procedures, or methods across the life cycle.
- Changes in people, structural, and task variables: The introduction of CASE technology will influence the roles or relationships of people and the tasks that they perform. The specific nature of these changes is not postulated beforehand, but left open to investigation in the assessment.

The PLC software development life cycle is expected to change due to application of the CASE tool. By using model-based analysis, testing may be done earlier in the life cycle by analysis of the logic in the functional description, prior to its translation into RLL code. The translation of the FD to RLL may be automated due to automatic code generation and this, in turn, is expected to encourage maintenance of the FD during the testing, commissioning, and troubleshooting phases of the life cycle. The expected life cycle is shown in Figure 7.1.

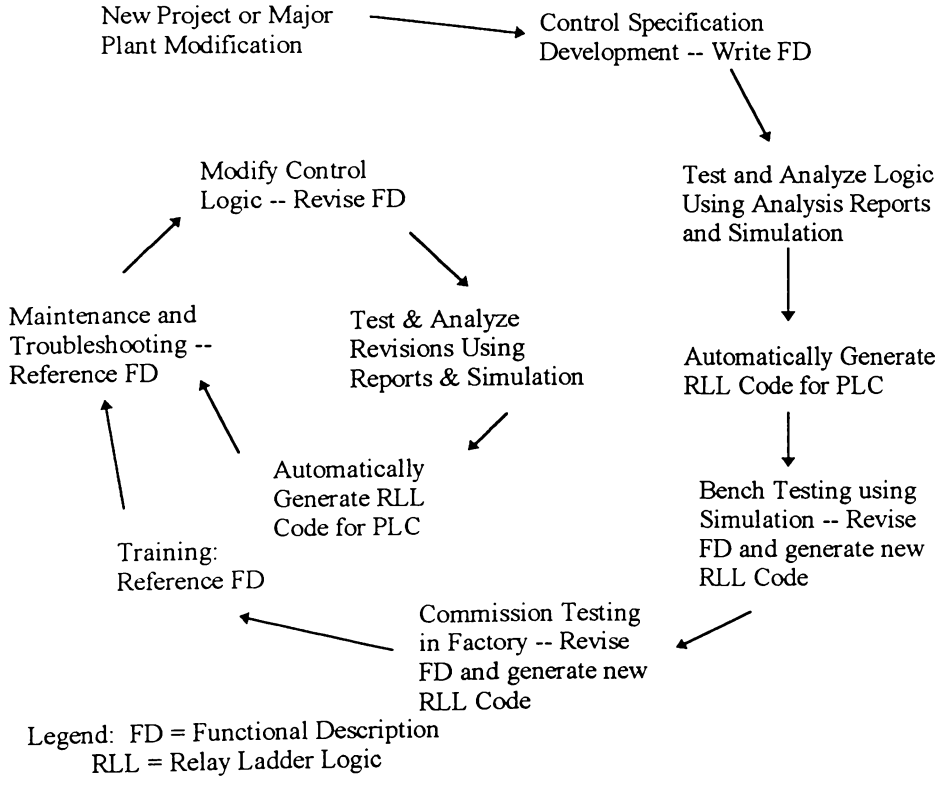


Figure 7.1. Expected Software Life Cycle

7.3 Users and Organisations Surveyed

The prototype CASE tool was introduced to a cross section of organisations and users involved in the PLC software development life cycle. The following five organisations were included:

- Anchor Products
- Two process automation consultant firms (Robinson Automation and Dench Automation)
- An automation software engineering firm (Computer Systems Engineering (NZ))
- PLC hardware and services vendor (Allen-Bradley (NZ))

Individuals included managers (one upper and five middle), two process engineers, two software engineers, and a plant operations engineer. The middle managers were all involved in overseeing some aspect of the control system development and most had prior experience in developing PLC code. The upper level manager was a general manager for the PLC vendor.

Some of the users were involved with the author over the course of the investigation and were familiar with the CASE tools while others were new to the project. Thus, to obtain a nominal baseline of knowledge of the CASE tools among all interviewees, a thorough, structured review and demonstration of the tools was presented to each interviewee. Each review and subsequent structured interview required about a half day per user.

7.4 Structured Interview Questionnaire

The structured interviews followed the tool review. A structured interview questionnaire was developed to guide this process and to obtain information relevant to the research questions.

An initial set of interview questions was derived from the research propositions. The questionnaire consisted of primary, open-ended, neutral questions. If more information on a given response was necessary, the primary question was followed up with probing, secondary questions. The use of primary, secondary, open-ended, and

neutral questions is reviewed in (Stewart, 1982). The initial questionnaire was tested in a pilot interview and found to contain excessive technical terminology and references to the dimensions of CASE tool functionality that appeared to the interviewee as irrelevant information.

Based upon the pilot interview experience, the questions were extensively revised to significantly reduce the amount of technical terminology and to eliminate extraneous information. The result was a set of questions grouped into the following five categories:

- *The impact of support for and enforcement of standards imposed by the CASE tools, in comparison to current methods.* Interviewees were asked to judge, in comparison to current methods, the impact of standardisation on the FD in their effort to create, test, and maintain the products (eg. FDs, RLL code) for which they are responsible and the potential influence on the quality (or other measures) of those products. Interviewees were also asked to judge the impact of standardisation upon others and upon the overall organisation. Finally, they were asked to identify any drawbacks to standardisation.
- *The effectiveness of domain-specific methods, languages, and rules in comparison to current methods.* Interviewees were asked to compare the domain-specific support offered by the methodology companion to their current tools (eg. word processor, vendor's RLL editor, etc.) and to judge the effort to use and learn the methodology companion and whether it fits their needs.
- *The use of the CASE tools themselves as compared to current methods.* These questions focused on the implementation of the prototype methodology companion. Interviewees were asked to compare it to their current tools and judge their effort to create, test, and maintain products for which they are responsible and the potential impact on the quality (or other measures) of those products. Next, they were asked to comment on the impact of the methodology companion on each of the following: the potential to facilitate reuse of software artefacts; the potential of automatic code

generation in each life cycle phase; the potential of the companion to influence long-term maintenance of the FD; and the potential effectiveness of the analysis and simulation tools in the CASE tool. A final question gave participants the opportunity to comment on any anticipated changes, such as tasks added or removed to their work, or changes in relationships with others, that may result from adoption of the CASE tool.

- *The potential of CASE tools on sharing and controlling of information compared with current methods.* Interviewees were asked to compare the prototype methodology companion to current methods and judge the impact of the CASE tool upon sharing of FDs within the organisation and the potential for controlling (eg. changing) the FDs.
- *Concluding remarks.* Participants were asked if, given a choice, they prefer the methodology companion or their current methods and to elaborate on their reasoning.

All interviews were recorded using both audio and video formats. To facilitate analysis of the content of the interviews, they were transcribed to text and then indexed. Following transcription, the responses were organised into the four categories of the expected outcomes: production, coordination, organisational, and people-structural-task. Appendix B presents the categorised interview responses with passages numbered for reference in the following discussion. The results shown in Appendix B are condensed from the original.

7.5 Results of the Interviews

Table 7.1 summarises the respondent's judgements concerning the potential of the methodology companion to improve current practice in the areas of production, coordination, and organisational support. A plus (+) indicates that the respondent expected that the companion would improve current practice, and a negative (-) indicates that it was not expected to improve current practice. The numbers are citations to the condensed interview transcripts in Appendix B.

	Production		Coordination		Organisational		
	Productivity	Quality	Control	Communication	Ease of Use	Learning	Standards
M/PLC	+ 1.12, 1.16, 3.3-3	+ 1.4, 1.14, 3.4	+ 2.5,2. 8	+ 2.7, 2.9	+ 3.6	+ 3.6	+ 3.1, 3.2-3.4
SM1	+ 1.26, 3.12	+ 1.26, 3.13	+ 2.14	+ 2.13, 3.13	+ 3.9	+ 3.10	+ 3.12-3.18
SM2	+ 1.33, 1.35, 3.22	+ 1.35	+ 2.15, 2.18		+ 3.24		+ 3.22
EM	+ 1.50, 3.25-3.26	+ 1.50, 3.28	+ 2.22	+ 3.26	+ 3.29		+ 3.25-3.26
SE1	+ 1.58, 3.30	+ 1.58, 3.31	+ 2.26, 2.27	+ 3.34	+ 3.35-3.36		+ 3.30
SE2	+ 1.68, 1.73, 3.38	+ 1.71, 3.39	+ 2.33	+ 3.40, 3.42-3.44	+ 3.47	+ 3.48	+ 3.38-3.45
PE1	+ 1.81	+ 3.51, 3.56		+ 3.52, 3.55	+ 3.54	+ 3.54	+ 3.50-3.52
PE2	+ 1.87	- 1.89	+ 2.37	+ 3.64	+ 3.66	+ 3.67	+ 3.64
OE	+ 1.97, 1.99, 3.70	+ 1.97	+ 2.40	+ 3.70	+ 3.74	+ 3.74	+ 3.66, 3.75
%	100	89	89	89	100	67	100

Legend: + respondent expected improvement over current practice.
- respondent expected no improvement over current practice.
Blank indicates no comments. Numbers refer to comments in Appendix B.
M/PLC: Managers from PLC vendor (general manager and two engineering managers)
SM1 and SM2: Managers of PLC and related software development
EM: Manager of Engineering at Anchor
SE1 and SE2: PLC software engineers
PE1 and PE2: Process engineers
OE: Plant engineering operations manager at Anchor

Table 7.1. Interview Results for CASE Tool Effectiveness.

7.5.1 Production

When asked to compare the methodology companion to their current tools and to judge their effort to create, test, and maintain products for which they are responsible, the respondents unanimously felt that either their own or the organisation's productivity would increase and eighty-nine percent of the respondents report that quality should improve. (See Table 7.1)

The expected improvements in both productivity and quality stem from the enforcement of standards by the methodology companion and the use of the simulation and report generation capabilities (model-based analysis) of the CASE tool. The standardisation of the FD language provides support in the area of representation, as there will be a single FD structure and language to learn, which is seen to especially benefit the recipients of FDs: the software engineers and plant operations staff. The

use of simulation and report generation provide support in the production sub-category of analysis. These capabilities are expected to result in better quality functional descriptions which in turn will improve productivity and quality throughout the life cycle. Automatic code generation was seen as a potentially less effective contributor in this area, and consequently the CASE tool provided the least improvement in the subcategory of automatic transformation of the FD into the RLL code. However, the standardisation of the FDs was identified as improving the transformation process, even in the absence of automatic code generation. The potential and limitations of model-based analysis and code generation are further discussed in Section 7.6.

Another aspect of support for production is the capability to reuse design or code artefacts. Several respondents expected that a set of FD components, representing control of certain plant functions such as pasteurisation and separation, could be developed and later applied in particular plants (1.99, 2.13, 2.24). It may also be possible to reuse a complete, existing FD, but this seemed less likely that the former approach (1.66).

7.5.2 Coordination

Interviewees were asked to compare the prototype methodology companion to current methods and judge the impact of the companion upon sharing of FDs within the organisation and the potential for controlling (eg. changing) the FDs. The respondents reported support for improvements in the areas of control and communication technology as a result of using the CASE tools and associated standards. As shown in Table 7.1, eighty-nine percent of respondents identified expected improvements in these two areas.

Respondents identified the history recording mechanism of the methodology companion as providing a very useful tool for controlling changes to the functional description (2.5, 2.8, 2.14, 2.15, 2.26, 2.27). Respondents felt that the history mechanism, with its capability of recording information at various levels of the FD hierarchy (2.14) could be used to record changes in PLC software for later entry into

the FD (2.5), to generate audit trails of changes in FDs (2.15, 2.26), and also as a revision control system (2.8).

Some additional potential areas of improvement identified by respondents are elaborated as follows. In the sub-category of coordination, enhanced communication amongst individuals involved in the software development process was identified as an expected consequence of the use of the methodology companion. This is attributed primarily to the enforcement of a standardised FD thus facilitating more and varied kinds of individuals to learn to understand the FD and then become involved in their development and maintenance. Communication amongst all groups is expected to become more effective (3.13, 3.40-3.44, 3.64).

If a standard set of reusable FD modules were developed (as discussed in Section 7.5.1), then these could be used as a starting point for development of FDs by either new or less experienced individuals (2.13). Furthermore, people who do not normally use FDs are expected to become familiar with the standard and then will be able to participate in the design process (2.7, 3.52, 3.55, 3.70). Finally, the summary reports that are produced by the FDA were identified as being the basis of a plant operations manual, and the support for the FD hierarchy in the methodology companion was seen as letting operators work at the most appropriate level of detail (1.37, 1.77, 3.26).

The author anticipated that the CASE tool would contribute to improvements in control by support for design maintenance; that is, the capacity of users to update the functional description and regenerate RLL code from it, rather than direct maintenance of the RLL code. This proposition was not supported by the respondents, primarily because of the methodology companion lacks the capability to generate code for incremental changes and to insert them into an operational PLC. This weakness is further addressed in Section 7.6 where the automatic code generation feature is discussed.

7.5.3 Organisational

The organisational category includes ways that CASE technology can provide an infrastructure to help support the areas of coordination and production. Infrastructure takes the form of support for standards and support for the particular application domain.

Respondents unanimously felt that the enforcement of standards would improve productivity in the development of control systems, and also the quality of those systems, in the overall organisation. In terms of the individual, the software engineers were seen as primary beneficiaries of standardised FDs, as this would eliminate time and effort needed to adapt to and understand the various FD formats present in the current system.

Individual process engineers, who would have to change to a standard FD language from their current methods, did not believe that their individual productivity or the quality of their FDs would improve, but they did concede that standards would be better for all other individuals in the organisation. One of the process engineers did believe that his productivity may improve due to the improved communication that would result from a standard FD (3.52, 3.55).

The enforcement of standards by the methodology companion is also seen as contributing to portability of people and their skills. Engineers should be able to more easily move from project to project when the FD has been standardised.

A cautionary note is that enforcement of standards could constrain innovation. Also, standards require maintenance, and they are costly in that there is usually an alternative method or product that is less costly than that required by the standard.

A sub-category of organisational support is functionality to help individuals use and understand the development environment. Interviewees were asked to compare the methodology companion to their current tools and judge their effort to create, test, and maintain products for which they are responsible and the potential impact on the quality (or other measures) of those products. The methodology companion was described as

being as easy or easier to use by all respondents, and it was reported by sixty-seven percent of users as easier to learn (some respondents did not comment on ease of learning). Domain-specific methods, the Windows interface, and the automated features of the methodology companion (such as supplying pick lists of plant items and operate/state keywords, automatic insertion of comments and checks, etc.) were identified as features that make the software easy to learn and use (4.7).

Several respondents identified the word processor as either being difficult to learn, or difficult to use to create a FD due to the difficulty of dealing with fonts, document layout, and other issues not related to batch manufacturing (3.35, 3.48, 3.66, 4.7). They felt that the methodology companion offered potential to relieve this burden. This finding supports the theory that generic tools, like the word processor, have a large cognitive distance which requires extensive specialisation to adapt them to particular domains. A consequence of this is that the domain specific methodology companion would allow FD writes to focus on the task of developing the control specification rather than the details of formatting a document with the word processor (1.34, 3.35).

Another factor contributing to ease-of-use is that the methodology companion is seen as providing a more integrated tool set (1.27, 1.67, 1.72). The methodology companion was described as centralising the functions of FD writing and PLC I/O database development that is currently done with separate tools.

An aspect related to integration of tools is that the FD repository was identified as providing a centralised database of FDs that serve to provide a single place to store, update, and backup these documents (1.72, 2.18, 2.22, 2.30, 2.33). This was seen as improving the maintenance of FDs and assuring that one is working with the most recent version of the FD.

7.5.4 People, Structure, Roles and Relationships

Interviewees were given the opportunity to comment on anticipated changes, such as tasks added or removed to their work, or changes in relationships with others, that may result from adoption of the methodology companion. Respondents identified

several ways that the methodology companion, and its support of standardisation, may permit changes in the tasks assigned to individuals (their roles) and may even impact the structure of an organisation.

The general manager of the PLC vendor anticipates that engineers can be given project management responsibility earlier in their careers, and can be given supervisory responsibility for third party contractors who have been trained in the standard, due to the automated assistance for conformance to standards (4.2). This would allow engineers to take on managerial responsibility earlier in their careers, and result in a more powerful organisation (4.3, 4.6).

The CASE tools and standards are expected to reduce the number of specialty functions in the organisation. For example, the process engineer's role may be to describe the process in general terms and people with expertise in PLC programming may finish the FDs (4.17, 4.19), the work of the process engineer and software engineer may overlap, and people may become more 'cross-skilled' (4.13, 4.14, 4.21). In essence, individuals may become less specialised and the number of speciality functions in the organisation may be reduced (4.19).

An important phenomena in New Zealand is downsizing and outsourcing, where companies reduce their in-house engineering staff and instead rely upon outside consulting firms for design expertise (1.8). A result of outsourcing is that there are fewer people in an organisation that can design or understand the processes within a factory (4.2). For example there are few good FD writers (4.23). The methodology companion may be used to leverage the expertise of these experienced FD writers. For example, less experienced people would use the methodology companion to develop a FD, with the assurance that it will conform to the standard structure, then an expert FD writer can be used to perform an audit or fine tune the FD. This will make more people available for the work of FD development (4.23, 4.25). The structure provided by the methodology companion, the automated aids such as lists of items in pull-down menus, and the fact that the methodology companion was seen as easier to learn and use than a

word processor were identified as supporting development and maintenance of FDs by a greater variety of people than current methods (3.14, 4.8, 4.18).

These role changes led several respondents to characterise the methodology companion as 'empowering' users by giving them more appropriate tools and also giving plant people a sense of 'ownership' over the FDs as they become involved with maintaining and even helping to develop FDs (3.14, 3.21, 3.69, 3.72, 4.8). This process is believed to lead to improvements in productivity when operations people can maintain the FD directly, rather than working through an engineer (4.18).

This change in roles may require the use of better skilled factory operations personnel. The plant operations engineer stated that the long-term goal would be to have very skilled people running the plant, as opposed to some of the current people who were hired for "his brawn rather than his brain," and that it is the better skilled people who will take ownership of FDs and be in a position to maintain them (3.69).

If automatic code generation is utilised to translate FDs to PLC code, the time required for coding will be reduced (4.8, 4.13) and thus software engineers may become more involved in FD development or may shift to development of other information systems (4.14, 4.11). A manager and a software engineer identified the possibility that software engineers may lose interest in their jobs if automatic code generation became a reality (4.11, 4.12, 4.14).

7.6 Appraisal of the Theoretical Framework

A hypothesis of this investigation is that CASE tools that incorporate domain-specific methods, support design maintenance, automatic code generation, and model-based analysis will be effective in addressing problems in the life cycle for manufacturing control software. The respondents judgements concerning these issues are summarised in Table 7.2, and are the topic of this section. As in Table 7.1, the plus (+) symbol indicates that benefits, in comparison to current methods, are expected by application of the idea, a minus (-) indicates that no improvement is expected, and numbers are citations to Appendix B.

	Companion Fits the Domain	Model-Based Analysis	Design Maintenance	Code Generation	Overall
M/PLC	+ 3.7	+ 1.1-1.4,1.14,1.15	- 2.6, 2.8	- 1.20	+ 1.21, 1.9-1.10
SM1	+ 3.20	+ 1.23, 1.27	+ 2.11, 3.21	+ 1.29, 1.31	+ 1.32
SM2	+ 1.40, 3.22-3.24	+ 1.48	- 2.16	+ 1.42, 1.46	+ 1.49
EM	+ 3.29	+ 1.53	- 2.21	- 1.56, 1.43	+ 1.54
SE1	+ 3.37	+ 1.63, 1.64	- 2.28	- 1.59	- 4.14
SE2	+ 3.49	+ 1.75, 1.76	+ 2.29-2.31	+ 1.69, 1.74, 2.24	+ 1.78
PE1	+ 3.54	- 1.80, 1.84	- 2.34-2.35	- 1.83	- 1.85
PE2	+ 3.68	- 1.93, 1.94			- 1.96
OE	+ 3.74	+ 1.97	+ 2.38		+ 1.98
%	100	78	33	33	67

Legend: + respondent indicates that the idea is feasible and should offer improvements over current practice
- respondent indicates that the idea is not feasible or may not offer improvements over current practice.
Blanks indicate no comments. Numbers refer to comments in Appendix B.
M/PLC: Managers from PLC vendor (general manager and two engineering managers)
SM1 and SM2: Managers of PLC and related software development
EM: Manager of Engineering at Anchor
SE1 and SE2: PLC software engineers
PE1 and PE2: Process engineers
OE: Plant engineering operations manager at Anchor

Table 7.2. Interview Results Concerning the Framework of Ideas.

7.6.1 Appraisal of Domain Specific Methods

All interviewees judged the methodology companion to be appropriate to the domain, as shown in the first column of Table 7.2. When compared to current methods the methodology companion is characterised as easier to use and learn (Section 7.5.3) and is also perceived to empower more people to become involved in the development of FDs (Section 7.5.4). These findings are discussed below.

A reason cited for ease-of-use of the methodology companion is that it eliminates the drudgery of configuring the word processor to correctly format the hard-copy document. Reduction in effort to use (specialise) a tool is a key characteristic of domain-specific methods.

The methodology companion is seen as empowering more people to become involved in the development and maintenance of FDs. Domain-specificity plays a central role in this regard due to the fact that the CASE tool incorporates the standard

batch control structure and language (3.23), and includes menus and pull-down pick lists of choices to guide users. Currently, the development of FDs is viewed as a somewhat mystical process and there are few people experienced in their development, but the methodology companion was judged as an aid in overcoming this barrier (3.55, 3.67).

Even within a particular type of industry, such as dairy processing, respondents identified domain specific methods as important. The PLC managers, who have seen many dairy plants across New Zealand, reported that methods differ dramatically across these organisations. The general manager made the following statement about these differences: "... (it) gets right down to some professional, intellectual things within engineers going back many years" (3.7). Furthermore, this manager noted the need for an organisation to differentiate itself from the competition in order to gain a strategic advantage (4.1). These findings suggest that organisations may find advantage in using CASE tools that are not only domain-specific but are organisation-specific. This would allow them to create tools that fit their professional practice as well as giving them the potential to gain competitive advantage.

7.6.2 Appraisal of Design Maintenance

The theory behind design maintenance and automatic code generation is that maintenance will focus on the specification, with the executable code automatically generated from the specification. A primary goal of the prototype methodology companion was to tightly couple the FD to the PLC code by using design maintenance and automatic code generation. The results of the survey revealed that only one third of respondents believed that design maintenance in the complete sense was feasible in the complete form. Reasons for this result are addressed below.

In the prototype methodology companion, the code generator is loosely coupled to the PLC due to production of PLC code in ASCII import/export format for the Allen-Bradley PLC5. (This was the only alternative available, as the codes and protocols to support on-line interaction with the PLC are proprietary.) This mode of interaction

with the PLC meant that the code generator could not be used to input code directly into the PLC. Instead the ASCII import code is written into an ASCII file by the code generator and is then read into the Allen-Bradley relay ladder logic editor (which runs on a personal computer) which in turn loads the code into the PLC. Essentially this is a batch mode operation. It could be used to make an initial software load of the PLC, but it could not be used to make on-line changes in the PLC directly from the code generator.

This mode of code generation limits the feasibility of design maintenance and automatic code generation in this domain for the reasons that follow. During the testing phase the process and software engineers work together to debug the FD logic and the RLL code. When they find a problem, either in the logic or the code, they modify the FD specification on paper and the RLL code on-line and continue on with the testing. To use the code generator to produce a new software load from the specification and to completely reload the PLC with the new code is too time consuming and thus would not be utilised. Thus, the close coupling between the design and the code, which was intended in theory, is lost. Further on in the life cycle, after the factory is commissioned, it operates continuously, all day, for weeks at a time. The PLC cannot be taken off-line to do a new program load, as is required to utilise the automatic code generator and the ASCII import method of code generation. Thus, the software engineer will make software updates on-line using the RLL editor (as in the current environment) and will not change the FD specification and re-generate code from it. Again, the close coupling between the FD and the code, intended by the theory of design maintenance and automatic code generation, will be lost.

Although design maintenance in the complete sense is probably not feasible in this domain, the methodology companion does offer the potential of improving maintenance of FDs, through the use of the history mechanism, as was discussed in Section 7.5.2.

7.6.3 Appraisal of Automatic Code Generation

The potential for the successful application of automatic code generation in this domain is closely related to that of design maintenance, and thus only one third of the respondents felt that it would be practical in the domain. However, several interesting findings emerged.

The research confirmed the 80/20 rule: respondents reported that approximately eighty percent of the code in an operational PLC is described by the FD, leaving about twenty percent of the code in a PLC is not specified therein. Information not specified in a FD is that not related directly to process control and includes requirements for reporting, the man-machine interface (MMI), background fault checking of plant devices, and system management code. Also, algorithms for control of analogue components, such as control loops, are not written in the FD. Since most of the above requirements are *not* in the control domain it can be concluded that the code for an operational PLC in this environment is *multi-domained*: it includes management reporting and operator interface functionality, as well as process control (1.62).

Another potential limitation of automatic generation of RLL code identified by respondents (if the software engineer is required to later maintain it), is that the software engineer will be required to learn the structure of code that he/she did not create. However, this problem is mitigated by the fact that this already occurs in practice when the maintenance programmer is different from the developer (1.43).

7.6.4 Appraisal of Model-Based Analysis

As is summarised in Table 7.2, practitioners expressed essentially universal support for the analysis capabilities available through the FDA. The simulation functionality, as well as the automatic generation of cross references, summaries, and the find and replace functions were seen as significant aids that could be applied throughout the software development life cycle.

Most respondents expressed strong support for the use of simulation to help evaluate the control logic in the FD. Support ranged from enthusiastic by higher level

management, who felt that it could be used to significantly improve quality, to weaker support amongst the process engineers (who would be the primary users of the it) whose primary reservation was that the time pressures imposed upon them would preclude the use of simulation. See Section 8.4.1 for additional discussion of this topic.

All respondents believed that the analysis reports produced by the methodology companion would help to improve productivity or quality.

7.6.5 Overall Acceptance

Participants were asked if, given a choice, they would prefer the CASE technology or their current methods and to elaborate on their reasoning. Two-thirds of the respondents (see Table 7.2) reported that they would prefer the methodology companion over their current methods. The reasons for preference of the CASE technology are that it structures and standardises the development of FDs, its ease of use, that it represents a step toward automatic code generation, and the feeling that the methodology companion is a better tool.

The two process engineers and one software engineer felt that, at present, they preferred their current methods. The software engineer's reason for preferring his current methods related to job interest -- he felt that the new system, with the automatic code generator in place, would make his job mundane. The process engineers agreed that the new system, with the standards that come with it, would be best for the organisation. However, one process engineer felt that at present the CASE tool did not generate hard copy output to an acceptable level of quality and the other process engineer stated that he preferred his current methods because that is what he has always used and, being human, did not want to change.

7.6.6 Appraisal of Life Cycle Changes

Figure 7.1 illustrated the anticipated changes in the life cycle as a result of application of the prototype CASE tool. However, because of the lack of capability to make on-line changes to an operational PLC's code, the anticipated model did not hold during the maintenance and troubleshooting phase. The problem is highlighted by the

boxed-in areas in Figure 7.2, where it was impossible to automatically generate incremental RLL code changes and load them into an operational PLC.

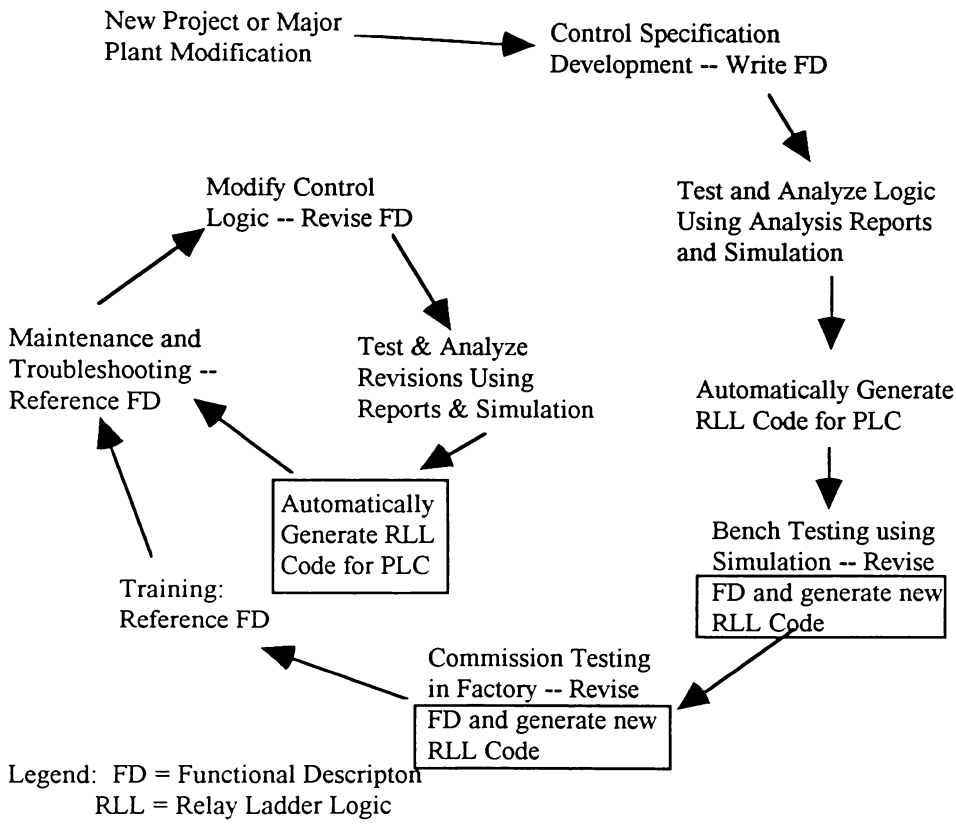


Figure 7.2. Enclosed Areas Indicate Problem Areas in the Anticipated Life Cycle.

Note that this life cycle reinforces the appraisal that design maintenance in the theoretical sense (maintenance of the specification and automatic generation of operational code from it) is most likely to be unsuccessful using the prototype CASE tool as implemented.

7.7 Other Benefits Identified by Respondents

One characteristic of action research is that unanticipated results typically emerge during the process of working with the participants. This section discusses such results.

7.7.1 Changes in Perceptions

The process of introduction of the CASE tool served to educate users about the need for standardisation in the software development life cycle. The general manager and the engineering manager of the PLC supplier felt that the CASE tool helped them

to identify variables in their current practice that could benefit from standardisation, such as the development of the PLC I/O data base and the data base for the operator interface system (1.11). A process engineer, who initially did not believe that the methodology companion would be a productivity aid, later changed his thinking when he realised that the standardisation of the functional descriptions would facilitate his communication with others in the organisation who are not normally used to working with FDs. He stated that this improved communication would significantly aid his productivity (3.52).

Participants also began to see the functional description and the control software as belonging to the organisation as a whole, and not to an individual or job-category. This is discussed in the next section.

7.7.2 Ownership of the Control System Becomes Organisation-Wide

The findings have shown that the methodology companion and its concomitant enforcement of standards provides: 1) A better integrated tool than current methods, thus centralising tasks formerly done using individual productivity tools as well as providing a centralised repository for storage of FDs; 2) That it should empower a wider range of individuals in the organisation to become involved in the process of FD development and maintenance; and 3) These individuals are expected to take ownership of the control system. The author believes that these developments may serve to give the organisation as a whole a sense of ownership and responsibility for the development and maintenance of the complete control system. Once the functional descriptions are seen as an organisation-wide resource, and not simply under the purview of the process engineer then new opportunities open up for their development and maintenance. This is analogous to the organisation-wide ownership of data that a database management system provides for an organisation.

7.7.3 CASE Tools Facilitate a Symbiotic Design Process

The transfer of ownership for the functional descriptions from the specialists (process engineers) to the organisation as a whole opens the opportunity for

collaboration amongst many more people in the organisation during the development of functional descriptions (4.23, 4.24). Others in the organisation no longer perceive the development of function descriptions as a "black box" and can become involved in their development (3.55). This synergism in turn is expected to lead to improvement in the development process.

This finding supports the Winograd's contention that software tools may offer a sophisticated medium of communication which may in turn open new conversation networks that give the organisation the ability to recognise and realise new possibilities (Winograd, 1986). It also contradicts Orlikowski's findings which concluded that the introduction of CASE tools into a large organisation contributed to more specialisation of roles in the organisation. In his study, the users of the CASE technology (analysts and programmers) were given little control over the development and implementation of the CASE tools, which was the domain of the technical specialists. As a result, the users felt a loss of power and autonomy, and at the same time resented the growing power of the technical specialists. This contrasts with the approach of this investigation in which the potential users of the CASE tool participated with the author in its design. This collaborative relationship may have contributed to the CASE tool's acceptance and its facilitation of less specialisation of roles, rather than more. Additional benefits of collaborative development are discussed further in the next section.

7.7.4 CASE Technology and Standards Support Outsourcing

The move in New Zealand toward outsourcing was discussed in Section 7.5.4. The general manager of the PLC vendor expressed the opinion that the standards enforced by the methodology companion would provide a more effective environment for working with outside people under contract (1.8). Cost savings would result from reduction in training costs, and more importantly, engineers could be given supervisory responsibility over outside contractors earlier in their careers if standards were used and enforced.

7.8 Summary

To assess the methodology companion and its underlying framework of ideas, expected outcomes were enunciated, the prototype was then introduced to practitioners in industry, and structured interviews were used to obtain their informed judgment as to the effectiveness of the CASE tool on practice.

The expected propositions were framed in terms of anticipated improvements in production, coordination, and organisational technologies. The results indicate that the CASE tool will yield improvements in productivity and quality in all three areas. Relating these improvements to the underlying theoretical framework indicates that domain specific methods and model-based analysis are the primary contributors to improvements. Design maintenance and automatic code generation are lesser contributors, due primarily to the need in this domain for on-line modification of the PLC code, which is not supported by the prototype CASE tool.

Changes in the perceptions of participants emerged during this cycle of the research process. The process led some participants to identify problems in their current practices. It also helped them to see that their individual productivity might improve as a result of the organisation-wide standards imposed by the methodology companion.

Benefits of the methodology companion or its concomitant enforcement of standards identified by respondents included centralisation of operations and ownership of functional descriptions, improved communication and a resulting symbiotic relationship that may develop across the organisation, and that CASE technology may provide assistance to organisations involved in outsourcing.

Chapter 8 Conclusion and Recommendations

This investigation endeavoured to test the hypothesis that computer aided software engineering (CASE) tools that are domain specific, incorporate model-based analysis and automatic code generation, and incorporate an appropriate methodology and standards will be effective in supporting the software development life cycle for manufacturing control software for programmable logic controllers (PLCs). It was postulated that effectiveness of the CASE tool would be revealed by improvements in the areas of production, coordination, and organisational support. Also, changes in people's roles, organisational structural, and task variables were expected as a result of the introduction of CASE technology.

A prototype methodology companion was designed, implemented, and introduced to industry practitioners. To support the development of the methodology companion, domain-specificity was defined in terms of cognitive distance, abstraction, and specialisation and then a method to guide the development of domain specific CASE tools was articulated. This method was applied within the domain of batch manufacturing control software to produce a prototype methodology companion.

To judge the effectiveness of the methodology companion, structured interviews with a cross section of practitioners were used to obtain practitioner's informed judgement concerning the potential and limitations of the CASE tool and its constituent framework of ideas.

This chapter summarises the findings of the investigation, as well as other results of the work and concludes with a discussion of possible future research directions.

8.1 Conclusions

This investigation led to the design and implementation of a unique CASE methodology companion to support the specification and implementation of batch control software for PLCs. To produce this tool, a representative methodology and language for the specification of batch control logic was identified in the literature and

also in practice at a large industrial user and this methodology and language were formalised and incorporated into the CASE methodology companion. The assessment of the tool by users in industry indicates that the domain-specific CASE tool will be accepted and will contribute to improvements in all three areas of production, coordination, and organisation support.

The assessment also served as an appraisal of concepts advocated in the literature as factors to be incorporated into automated support aids for the software development life cycle. These include domain-specific methods, design maintenance, automatic code generation, and model-based analysis.

In this investigation, domain specific methods were seen to contribute strongly to the ease of learning, ease of use, and acceptance of the CASE tools by the user. Minimisation of the cognitive distance of the CASE tool by incorporating a domain-specific methodology and language into the CASE tool is viewed as a prime factor in the acceptance of the tool.

Model-based analysis is seen by the users as providing a significant aid in improving quality of the software specification. The availability of automated design aids, such as summaries and cross references, and simulation of the specification are seen as innovative contributions in the industry. A mitigating factor that may work against the use of simulation is the limited time available for the specification phase of the life cycle. In the interviews, both process engineers indicated that they believed that they would not have time available to devote to simulation. Thus, the potential of model-based analysis may not be realisable unless more time is allocated to the specification phase of the life cycle to permit the use of model-based analysis. See also Section 8.4.1.

The potential of design maintenance and automatic code generation is compromised by the inability to generate code incrementally and insert it into an operational PLC. Thus, for the potential of design maintenance to be realisable, CASE tools must incorporate functionality to support the testing and maintenance phases of the life cycle in a particular domain-specific manner.

In addition to the findings summarised above, the other results emerged from the investigation. These are discussed in the remaining sections.

8.2 Additional Results of the Investigation

Other contributions of this investigation include a definition of domain-specific methods that allows this concept to be operationalised, and, based upon this definition, an approach to guide the development of domain-specific CASE tools is described. These are discussed below.

Domain-specific methods are defined in terms of abstraction, specialisation, and cognitive distance. A domain-specific CASE tool is one that is designed to minimise cognitive distance by reducing the amount of specialisation necessary for a user to apply the tool. Effort to specialise the tool is minimised by maximising the amount of information embedded into the fixed and hidden parts of the CASE tool's abstraction, while providing a variable abstraction that is wide enough to accommodate the desired domain and to allow enough flexibility so as to not constrain innovation and creativity. This definition is based upon studies of software reuse technologies (Krueger, 1992).

The approach to guide the design of domain-specific CASE tools involves the following steps: 1) Identify the domain by performing a domain analysis to identify methodologies, models (abstractions), and languages that are appropriate to the domain, including appropriate abstraction specifications and abstraction realisations; 2) Decide what aspects of the domain should be included in the CASE tool. The domain width should be only as wide as necessary to accommodate user needs; 3) Identify the abstraction (or model) presented to the user. Choose a specification language that is acceptable to the user and also succinct and expressive. It may be necessary to provide a way to allow some functionality to be programmed directly into the implementation technology (ie, the 80/20 rule) since all of the requirements may not be covered by a single domain; 4) Decide what aspects of the abstraction are under the control of the user (variable part) and which parts are fixed; 5) Define the way the user specifies the system under development. Options include textual specification languages, templates,

graphical diagrams, interactive menu-driven dialogue, and structure-oriented editing; 6) Define the products of the tool, and 7) Implement the CASE tool. This approach is based upon work in domain analysis and application generators (Cleaveland, 1988).

The next section reviews the research approach and summarises the author's experience with it.

8.3 Characteristics and Benefits of the Research Approach

This section reviews the author's views of benefits of the action research approach that emerged during this investigation.

8.3.1 A Cyclic Process

Action research is characterised as a cyclic process of diagnosing, planning, action taking, evaluation of results, and specification of findings. In this investigation, three major cycles, and their results, can be identified as follows: 1) The specification of the roles of the researcher and the client organisation and the initial domain analysis that led to the articulation of an approach to guide the development of the methodology companion; 2) The subsequent collaborative relationship between the researcher and the practitioners that resulted in the design and implementation of the prototype methodology companion; and 3) The assessment of the methodology companion by practitioners which led to conclusions concerning the potential of it and also resulted in an assessment of the framework of ideas upon which the companion is based.

Benefits of the action research approach are discussed below.

8.3.2 A Learning Process

The use of action research served to educate the author about the manufacturing domain and the unique requirements of that domain. The availability of practitioners who could explain the operation of PLCs, supply example design and coding

documents, and in the end to assess the prototype CASE tool were critical to the conduct of the investigation. The fact that the prototype CASE tool is acceptable to the users and may be adopted in practice (subject to management decision) differentiates this project from many in software engineering where the results of the research have not transferred to industry (Potts, 1993).

Another aspect of action research is the change in understanding and perceptions on the part of the clients that result from their involvement in the investigation. In the process of explaining their methods and developing sample model documents for the investigator, the practitioners were led to identify and confront their need to standardise their FD specification language. Ambiguities in the language were uncovered as practitioners described the documents to the investigator, and management was led to use the project as a springboard for developing standards for the FD specification. In the process of analysing the RLL code and its relationship to the FD, the users identified the need to more fully document some of the informal practices in the coding phase of the life cycle. Also, in discussions of the potential of the CASE tool to contribute to standardisation and design maintenance, problems caused by the lack of standardisation and design maintenance were highlighted.

8.3.3 Incorporation of Management Decisions into CASE Tools

The collaborative research approach resulted in development of a tool that incorporated Anchor management's desires for a way to enforce standardisation of functional descriptions. It was pointed out in Chapter 5 that system designers (in this case the author) wield significant power because of the potential effect that a computer system may have on an organisation (Winograd, 1986). By working in a collaborative fashion with practitioners, design decisions such as the form of the tool (methodology companion vs. a FD compiler) and the specification language (traditional batch control language vs. a new language) were made that are acceptable to the users and also resulted in a tool that may support significant structural changes in the organisation, as was discussed above.

8.3.4 Collaboration with Practitioners Reduces Barriers to CASE Tool Acceptance

A reason that the CASE technology in this investigation was accepted by users may be due to the fact that they were involved in its development. The interview results revealed no strongly negative reaction on the part of practitioners against potential adoption of the methodology companion. This contrasts with studies of unsuccessful CASE tool adoption that revealed that tools may not be adopted due to the use of power-coercive strategy of CASE implementation, where the use of CASE technology is imposed upon the users by management (McLendon, 1992, Orlikowski, 1989). For example, Orlikowski's study reported behaviour including territorialism, resentment, and even rebellion among team members who felt that their job functions were threatened by the imposition of CASE technology. No evidence of such behaviour was identified in the investigation described herein, which may in part be attributed to the action research approach, which emphasises a normative-educational approach, rather than a power-coercive approach to change. (The normative-educational approach (Bennis, 1985) emphasises extensive group inclusions to assist in the goal of gaining acceptance and changing attitudes.) Thus, the action research approach may facilitate adoption of CASE technology.

8.4 Implications of Findings for Effective CASE Tool Use in the Domain

This section discusses some implications of the findings with respect to model-based analysis and design maintenance.

8.4.1 Effective Use of Model-Based Analysis may Require Changes in the System Life-Cycle

The process engineers, who are currently the authors of functional descriptions, reported that simulation before code generation may not be feasible due to the extremely short time allocated for the development of FDs. This constraint is partially attributed to the domain, where engineer's may have only a few weeks of notice that they have budget approval to do a job (See Appendix B, 1.7). Thus the use of

simulation, and model-based analysis in general, may require changes in the life cycle of factory development that will allow more time for FD development.

8.4.2 Design Maintenance Requires Additional Tool Support for Debugging and Maintenance

One finding of this investigation is that design maintenance and automatic code generation, in its ultimate form as envisioned by researchers, is likely to be unsuccessful using the prototype methodology companion in this domain. This section discusses alternatives to design maintenance and automatic code generation that emerged from the structured interview process.

An alternative to modification of the RLL code on-line, which still uses automatic code generation during the development phase and supports design maintenance, is to generate the initial PLC code, permit on-line maintenance of the RLL code using the PLC editor, and then, when necessary, use inverse (or reverse) engineering to show where the FD has been modified. Inverse engineering is the process of deriving a high level specification of a program from the code.

In the absence of an initial high level specification, this process requires guidance from a human, as has been reported in the general sense in and in the case of inverse engineering of ladder logic in particular (Ward, 1989; Falcione, 1993). This is because the structure of ladder logic code typically bears little resemblance to the original specification. However, if the RLL code was annotated with marks, dubbed *bookmarks*, when the code is initially generated by the methodology companion, then an automated tool, called a *maintenance companion*, could be provided that could compare PLC code running in a PLC with the original annotated code, and then generate a report showing were the RLL code had changed. This report could map the RLL changes to the FD, and thus inform the user in which selection, or possibly in which step, the operational RLL code differed from the specification. Respondents to the interviews felt that this idea held promise for improving the maintenance phase of the life cycle. (See 2.10, 2.12, 2.17, 2.19, and 2.23 in Appendix B.)

Alternatives to generation of RLL code were identified by users. One alternative is to modify the specification, generate code for the modified portion of the system, display it, and allow the software engineer to use this as a guideline for making the on line modifications. Another alternative to automatic PLC code generation, but one with a potentially high return in terms of productivity is to replace the automatic code generator with a CASE tool that generates much of the PLC's data table, message database, and supplies reports that aid in the production of the PLC code. As was reported in Chapter 4, 12% of the total effort (25% of the coding effort) is devoted to production of the data table. Much of the information for the data table can be extracted or generated automatically from the functional description and the final data table could be produced in either report format or in an appropriate machine readable format for use in other tools such as the PLC vendor's RLL editor. Also, the plant simulation software used during the bench testing phase requires preparation of code that simulates the plant. This code could may be produced automatically by the methodology companion. (See Appendix B, 1.56.)

Another suggestion made by a respondent was to provide an alternate view of the functional description, such as a function chart or control chart view. Although this may prove difficult using the current FD structure (where steps start in sequence but are active concurrently), an alternative view may be useful for helping the users to migrate to a new specification language. (See Appendix B, 2.3.)

8.4.3 An Extended CASE Tool Architecture for Manufacturing Control

This research revealed that the theory of design maintenance and automatic code generation must be extended to encompass the aspect of debugging and maintenance, and the debugging facility should be at a level appropriate to the domain. For the batch process manufacturing domain in particular, the environment must support the capacity to modify the control code on-line using incremental automatic code generation and loading of the incremental code changes into an operational PLC. This necessitates addition of a new module to the generic case tool architecture. The module, shown in bold, is illustrated in Figure 7.1, and is called a *maintainer's assistant*.

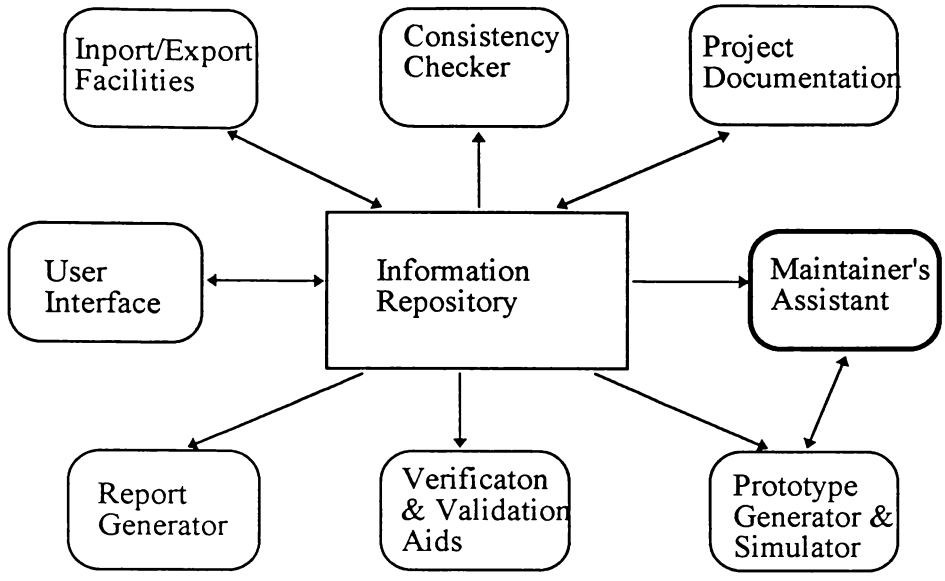


Figure 8.1. Extended CASE Architecture to Support Maintenance and Debugging.

The maintainer's assistant is conceptualised as supporting the maintenance and debugging methods particular to the target domain of the CASE tool. For batch process control, the maintainer's assistant would support incremental simulation, compilation, and up-loading of RLL code into the PLC. Additionally, the maintainer's assistant should provide the same capabilities as vendor-supplied RLL programming environments, such as the ability to monitor execution of the code in the PLC using animation, preferably of the specification. Design and development of this module is a significant opportunity for continued research.

8.5 Directions for Future Work

The prototype CASE tool may be developed to industrial standard so that it can be applied in production and the long-term effects of the CASE tool can be studied. This is best carried out by a continuation of the action research partnership, where the industry partner completes development of the CASE tool and the academic partner concentrates on studying the effectiveness of the tool.

The potential for model-based analysis to improve the quality of specifications appears to be significant. The functionality of this feature in the prototype could be significantly enhanced by adding support for writing simulation scripts that can then be

used for repeated testing, such as regression testing. Also, the simulator could be modified to include generation of test conditions, such as random changes of state of plant items in order to test unanticipated events on the control logic.

A limitation of the model-based analysis in the prototype is that it models only the logic of the FD but does not simulate the actual operator interface nor does it imitate the physical characteristics of a factory, such as verifying capacities, flow rates, or other mechanical operations. Coupling the FD simulation with an operator interface, or possibly with the plant and instrumentation diagram (P&ID), may be beneficial. Also the addition of rules to the dictionaries that describe the physical characteristics of plant items, such as capacities or other constraints, may then permit analysis of the appropriateness of certain logic in the FD, or the P&ID diagram could be automatically analysed for consistent interconnection of plant devices.

This investigation has identified several conjectures concerning the potential for CASE technology to change user's roles and relationships within an organisation which warrant further investigation. Conjectures are:

- By centralising the storage and maintenance of design specifications (FDs), CASE technology will facilitate a sense of organisation-wide ownership of the system.
- Domain-specific CASE technology facilitates communication which will provide a symbiotic relationship amongst individuals in the organisation, contributing to improvements in quality and productivity.
- Cooperative development of CASE tools reduces barriers to CASE tool acceptance.
- Incorporation of management policy into the tools may facilitate organisational changes, such as support for outsourcing and changes in responsibilities given to individuals.

These conjectures may be explored using additional action research studies.

Further research is needed to develop automated aids to support the debugging, maintenance, and troubleshooting phase of the life cycle if design maintenance and

automatic generation of code from the specification is to become a reality. Practitioners cannot be expected to regenerate entire applications in what is essentially a "batch" mode of operation when minor changes are made to the specification. The CASE tool must either support incremental code generation or some form of inverse (also called reverse) engineering to keep the specification current with the operational software. Furthermore, appropriate debugging technology is required, such as the ability to monitor execution of the code by animating the specification.

Manufacturing control systems incorporate multiple domains. In the case of this study, 80% of the PLC's code relates to the domain of batch control logic, but about 20% of the code is for system management, the man-machine interface, reporting, and overall structuring of the code. The suggested approach to handling the 20% of the domain outside the model is to allow the user to directly code this software. Other approaches may be possible, such as use of a second domain-specific CASE tool, or a combination of domain models within a single CASE tool.

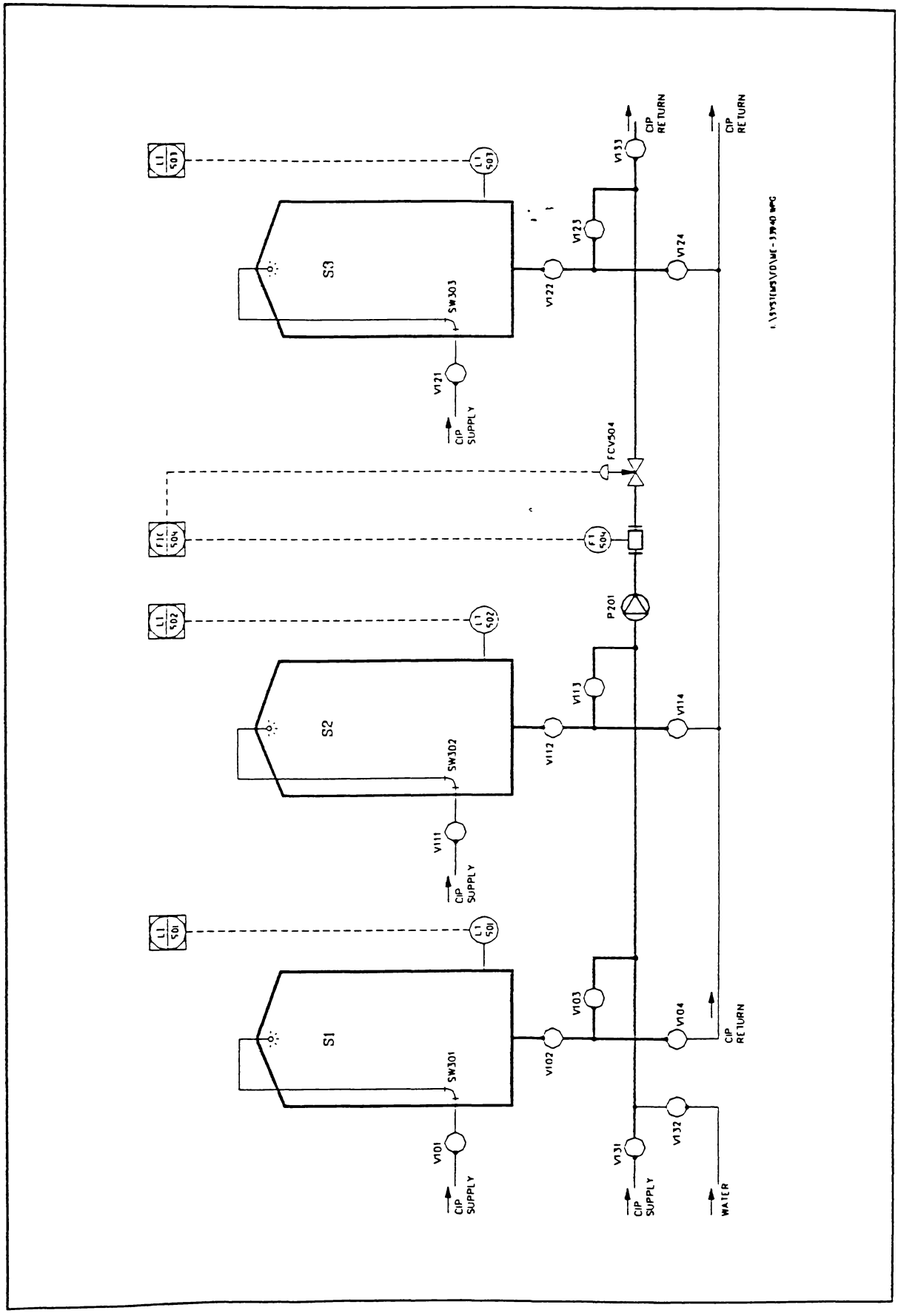
Ladder logic remains a low-level programming language which is difficult to structure and maintain. CASE tools that automatically generate ladder logic from a higher level specification are one step in an evolution away from the use of ladder logic as the primary PLC programming language. Other ways to facilitate a migration away from ladder logic are that the methodology companion could provide multiple views of the specification: one using the traditional batch language, another being a graphical representation. However, using the view-point of control employed by the practitioners in this study, where steps start in sequence but then continue to execute in parallel, alternative views using Grafcet or finite state diagrams are not practical. Application of one of these methods to the domain would require that the users (Anchor) adopt a different way of thinking about their control systems. Alternatively, research into modifications to Grafcet or new graphical specification methods appropriate to their methods may be in order.

Appendix A Sample Functional Description

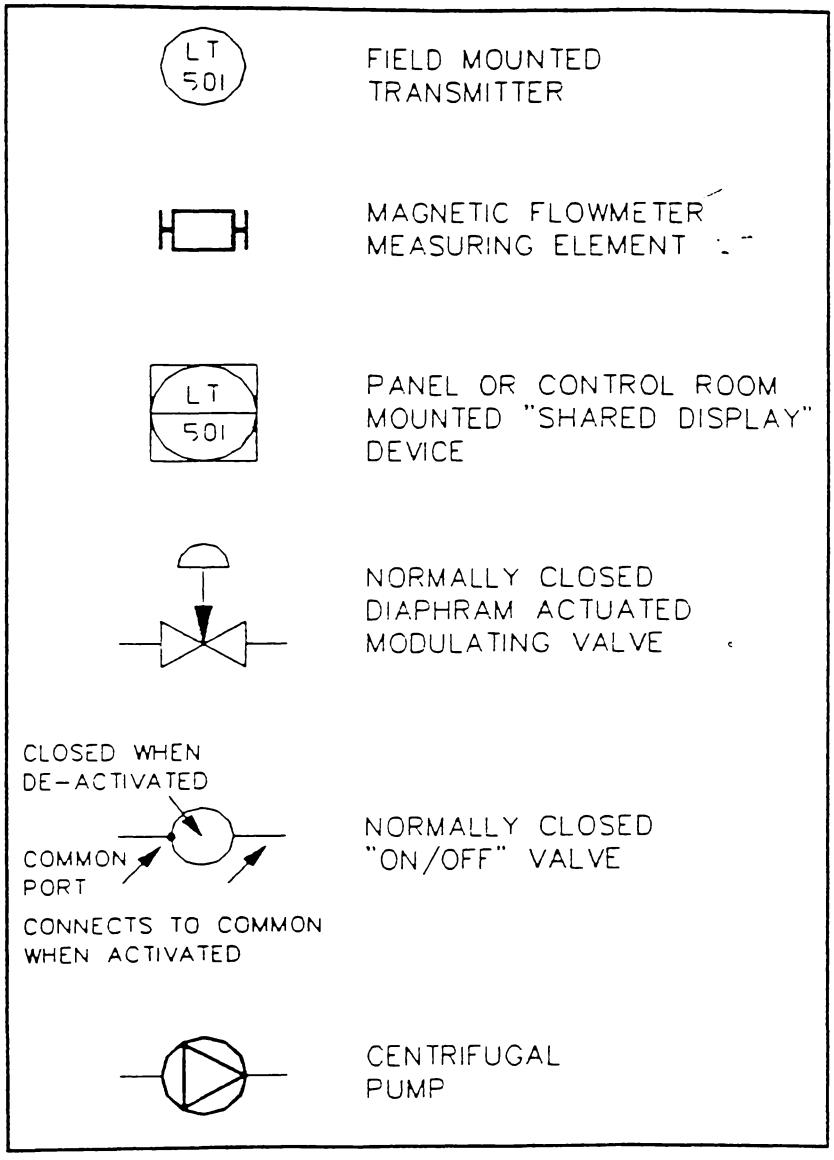
Table of Contents

1. Schematic
    1. Process & Instrumentation Schematic
    2. Legend of Symbols used
2. Process Description
3. Automation Database
    1. Real I/O
4. Function Description
    1. Conventions used in description
    2. List of Selections & Operator Entry Commands
    3. Selections Description
    4. Software Tags used
        Timers;
        Flags;
        Analogue Controller Setpoints
        Analogue Switchpoints
        Messages
    5. Reporting Requirements
    6. Operator Interface Requirements
    7. Comments & Assumptions

SECTION 1 : 1 SCHEMATIC DRAWING



SECTION 1 : 2 SCHEMATIC SYMBOLS



SECTION 2 : PROCESS DESCRIPTION

1. Milk is transferred from Silo S1 to Silo S2 at a fixed flow rate.
2. If the level in the source silo is approaching empty, or the level in the destination silo is approaching full, then an alarm is provided for operators.
3. If the source silo empties, or the destination silo becomes full, then a fixed volume of water is transferred to the destination silo, with the control valve in a fixed position. (There is sufficient free volume in the destination silo to take the flush water volume.)
4. The process can be started or stopped at any time, with or without a shutdown water flush.
5. Water in the line at the start of the process is allowed into the destination silo.
6. The process should be interlocked to prevent mixing of milk with CIP solutions.
7. A remote indication lamp that will indicate process in operation (steady indication); or process in fault (flash indication).

SECTION 3 : AUTOMATION DATABASE (REAL I/O)

Tag -	Description	I/O Type			
		DI	DO	AI	AO
V101	Silo S1 CIP Supply Valve	2	1	0	0
V102	Silo S1 Gullet Valve	2	1	0	0
V103	Silo S1 Process Outlet Valve	2	1	0	0
V104	Silo S1 CIP Return Valve	2	1	0	0
V111	Silo S2 CIP Supply Valve	2	1	0	0
V112	Silo S2 Gullet Valve	2	1	0	0
V113	Silo S2 CIP Process Outlet Valve	2	1	0	0
V114	Silo S2 CIP Return Valve	2	1	0	0
V121	Transfer Line CIP Supply Valve	2	1	0	0
V122	Transfer Line Water Supply Valve	2	1	0	0
V123	Transfer Line CIP Return Valve	2	1	0	0
P201	Transfer Pump	1	1	0	0
SW301	Silo S1 CIP Supply Gateswitch	1	0	0	0
SW302	Silo S2 CIP Supply Gateswitch	1	0	0	0
IN401	Remote Indication Lamp	0	1	0	0
LT501	Silo S1 Level	0	0	1	0
LT502	Silo S2 Level	0	0	1	0
LT503	Transfer Line Flow	0	0	1	0
CV504	Transfer Line Control Valve	0	0	0	1

SECTION 4 : FUNCTION DESCRIPTION

4.1. Conventions Used In Description

4.1.1 Selections

4.1.1.1 Each selection is initiated, either "selected" by an operator or automatically started by other selections.

4.1.1.2 Each selection has a series of associated steps, and conditions if necessary which cancel the selection.

4.1.1.3 The status of each selection should be displayed on the operator interface:

1. The selection number eg 101
2. The name of the selection eg "Milk Transfer to Silo S2"
3. The highest number step associated with the selection that is currently active
4. The description of that step eg "Open Route"
5. A single fault message, that displays any fault associated with that selection.

4.1.2 Selections

4.1.2.1 All Steps have *preconditions*, and may have *operates* and *checks*.:

1. Preconditions. Conditions that must be true for the Step to be activated. These conditions may be events (ie achieved instantaneously); or alternatively the conditions are monitored (ie the conditions must be continuously maintained)
2. Operates. Actions that are performed if the Step is active. All items are operated simultaneously, unless otherwise specified.
3. Checks. If the step is active, a series of statements are checked. If any statement is false, there is deemed to be a fault with the checks associated with that step. The particular statement that is fault is required to be displayed as a fault message on the operator interface screen. One fault message for each selection should be able to be displayed on the operator interface screen at any time. Where more than one fault exists, the priority for fault display shall be in the order listed in the selection.

4.1.2.2 All steps are independent. More than one step of a selection can therefore be (and usually is) active. The active steps of a selection are also therefore not necessarily sequential. The preconditions of a step define whether that step is active.

4.1.3 Digital Control

4.1.3.1 The logic associated with the control of digital outputs is continuously monitored and the output status table updated.

Digital outputs are controlled by logic associated with active steps. If the preconditions for that step are not true, the step is deactivated and the control of digital items from that step ceases. Therefore no explicit instruction to deenergise a digital output is required.

If a digital output is latched on, the output remains energised until the output is unlatched, regardless of the step logic. The conditions to latch the output; and the conditions to unlatch the output are therefore both detailed.

4.1.4 Analogue Control

4.1.4.1 Instruction for analogue control, for example

1. setting automatic setpoints;
2. setting control valve positions;

are all one shot actions.

This will allow these to be independently controlled from the operator interface.

4.2. List of Selections Used

Selections	Operator Entry Commands
101 Milk Transfer	[101 Transfer Start]
	[101 Shutdown Start]
	[101 Transfer Cancel]
102 Milk Transfer Level Alarm	[102 Alarm Cancel]

101. MILK TRANSFER

Step 101/0
The first step is a pre initiation step, that allows fault display of interlocks that are preventing the selection from being initiated.
If any CIP circuit is using part of the process transfer route, the selection should not be able to be initiated.
If such fault exists the selection will self cancel after 20 seconds, during which time the operator is reminded of the CIP operation.

STEP 101/0 (START)

Preconditions

1. Operator Action — event; operator interface [101/Start]
2. Step 101/1 Not Active .And. — monitor; step not reqd once selection starte
3. Timer 101/0 Not Complete — monitor; initial fault

Operates

If Step 101/1 is not active before the timer is done, the selection is cancelled. The timer therefore controls the time a fault message is displayed to the operator explaining why the selection will not run.

1. If Step 101/1 Not Active then: — approx. 20 secs; refer Step preconditions.
   1. Start timer 101/0

Checks

1. Step 201/1 Not Active — CIP of silo S1
2. Step 202/1 Not Active — CIP of silo S2
3. Step 203/1 Not Active — CIP of transfer line

STEP 101/1 (INITIATE)

Preconditions

1. Step 101/0 Active .And. — event; selection started
2. No faults Step 101/0 Checks — CIP interlocks ok
3. Step 101/9 Control — monitor; selection cancel

Operates

1. Flash Energ IN401 — The remote indication lamp provides a flash indication as soon as the selection is started

(Energ for 0.5 secs Deenerg for 0.5 secs)

If there are no faults with the selection, the selection will be running (refer Step 101/) which provides a steady indication.
Initialise the position of the transfer control valve.

2. If (Step 101/3 Not Active .And. — process; controller in auto
   Step 101/5 Not Active) then — shutdown; controller output fixed

1. Set FC503 Manual OP1 — controller output zero; close control valve
   Controller FC503 — transfer line flow controller
   Controller Input : FT503 — transfer line flow
   Controller Output: CV504 — transfer line control valve

Provide a fault message display (alarm) if source silo is nearly empty, or destination silo is nearly full

3. If Level LT501 < L3 then: — silo S1 low level; approx 10%
   1. Start (One Short) Selection 102 — Alarm Selection; generate fault display

4. If Level LT502 > L2 then: — silo S2 high level; approx 85%
   1. Start (One Shot)Selection 102 — Alarm Selection; generate fault display

Checks

1. V104 Deenerg (Closed) — silo S1 CIP return valve
2. V111 Deenerg (Closed) — silo S2 CIP supply valve
3. V114 Deenerg (Closed) — silo S2 CIP return valve
4. V121 Deenerg (Closed) — transfer lines CIP supply valve
5. V123 Deenerg (Closed) — transfer lines CIP return valve
6. If Step 101/5 Not Active then: — if shutdown not initiated, transfer lines wate supply
   1. V122 Deenerg (Closed)

STEP 101/2 (OPEN ROUTE)

Step 101/2
If all valves on the transfer route that should be closed, are checked to be in the correct position, the transfer route is able to be opened.

Preconditions

1. Step 101/1 Active .And. — monitor; selection started
2. No faults Step 101/1 Checks — monitor; CIP interlocks ok
3. Step 101/5 Not Active — monitor; shutdown initiated

Operates

1. Energ (Open) V102 — silo S1 gullet valve
2. Energ (Open) V103 — silo S1 process outlet valve
3. Energ (Open) V112 — silo S2 gullet valve
4. Energ (Open) V113 — silo S2 process inlet valve

Checks

1. V102 Energ (Open) — silo S1 gullet valve
2. V103 Energ (Open) — silo S1 process outlet valve
3. V112 Energ (open) — silo S2 gullet valve
4. V113 Energ (Open) — silo S2 process inlet valve

STEP 101/3 (START TRANSFER)

Step 101/3
If the transfer route is confirmed open, the transfer pump is able to be operated, starting the transfer.

Preconditions

1. Step 101/2 Active .And. — monitor; route opened
2. No faults Step 101/2 Checks — monitor; route confirmed open

Operates

1. Start P201 — transfer pump

2. Set FC503 Auto SP1 — flow controller automatic
   Controller FC503 — transfer line flow controller
   Controller Input: FT503 — transfer line flow
   Controller Output: CV504 — transfer line control valve

Checks

1. P201 Running — transfer pump

STEP 101/4 (RUNNING)

Step 101/4
The main purpose of the step is to provide a "running" indication to the operators; indicating transfer in progress with no faults.
Conditions for initiating shutdown automatically are also continuously monitored.

Preconditions

1. Step 101/3 Active .And. — monitor; pump started
2. No faults Step 101/3 Checks — monitor; pump running

Operates

Shutdown is initiated if the source silo is empty or the destination silo is full.
The source silo is deemed empty if the level is low, and the transfer flow is low (ensuring the silo si completely drained).
The destination silo si deemed full if the level is above a high recipe switch point - which allows some margin of safety in filling.
A timer is started to ensure the shutdown conditions are not instantaneous, and therefore not false.

1. If (Level LT501 < L4 .And. — silo S1 low level; approx 3%
   Flow FT503 < F1) .Or. — low flow; approx 5% full flow
   Level LT502 > L1 then: — silo S2 high level; approx 95%
   1. Start Timer 101/3 — approx. 15 secs; initiates shutdown
2. Energ IN401 — The remote indication lamp is continuously energised to indicate operation with no faults.

Checks

STEP 101/5 (S'DOWN INITIATE)

Step 101/5
Shutdown is initiated at operator discretion; or automatically.
When shutdown is initiated, Step 101/2 is cancelled; which automatically closes the product transfer route.
The water flush route needs to be similarly checked.

Preconditions

1. Step 101/1 Active .And. — monitor; selection started
2. (Operator Action .Or. — event; operator interface [101/Shutdown]
3. Timer 101/4 Done) — event; source silo empty; or destination full

Operates

1. Set FC503 Manual OP2 — flow controller manual; fixed output positio
   Controller FC503 — transfer line flow controller
   Controller Input: FT503 — transfer line flow
   Controller Output: CV504 — transfer line control valve
2. Start Flow Totaliser FY503 — Flow totaliser is reset; and starts accumulating (Accumulator by software integration)

Checks

1. V102 Deenerg (Closed) — silo S1 gullet valve
2. V103 Deenerg (Closed) — silo S1 process output valve

STEP 101/6 (OPEN ROUTE)

Step 101/6
If all valves on the transfer route that should be closed, are checked to be in the correct position, the (water flush shutdown) transfer route is able to be opened

Preconditions

1. Step 101/5 Active .And. — monitor; selection started
2. No faults Step 101/1 Checks — monitor; CIP interlocks ok
3. No faults Step 101/5 Checks — monitor; source silo output closed

Operates

1. Energ (Open) V122 — transfer line water supply valve
2. Energ (Open) V112 — silo S2 gullet valve
3. Energ (Open) V113 — silo S2 process inlet valve

Checks

1. V122 Energ (Open) — transfer line water supply valve
2. V112 Energ (Open) — silo S2 gullet valve
3. V113 Energ (Open) — silo S2 process inlet valve

STEP 101/7 (START TRANSFER)

Step 101/7
If the transfer route is confirmed open, the transfer pump is able to be operated, starting the transfer.

Preconditions

1. Step 101/6 Active .And. — monitor; route opened
2. No faults Step 101/5 Checks — monitor; route confirmed open

Operates

1. Start P201 — transfer pump

Checks

1. P201 Running — transfer pump

STEP 101/8 (RUNNING)

Step 101/8

The main purpose of the step is to provide a "running" indication to the operators; indicating (shutdown) transfer in progress with no faults.

Preconditions
1. Step 101/7 Active .And.
2. No faults Step 101/7 Checks

monitor; pump started
monitor; pump running

Operates

1. Flash IN401

The remote indication lamp is continuously energised to indicate operation with no faults.
remote indication lamp

Checks

STEP 101/9 (CANCEL)

Step 101/9
The selection is cancelled by the operator at any time; or automatically cancels at the completion of the shutdown water flush
If the selection is cancelled by the operator before shutdown is initiated, the transfer line will not receive a water flush. If the selection is cancelled during a shutdown sequence, the water flush will be immediately terminated.

Preconditions
1. Step 101/1 Active .And.
2. (Operator Action .Or.
3. FY503 > F2)

monitor; selection initiated
event; operator interface [101/Cancel]
event; flow totaliser > preset shutdown flus volume.

Operates
1. Set FC503 Manual OP1
   Controller FC503
   Controller Input: FT503
   Controller Output: CV504

The analogue controller is reset controller output zero; close control valve
transfer line flow controller
transfer line flow
transfer line control valve

2. Delay one scan; then
   1. Cancel Selection 101

cancel selection;l refer Step 101/1 preconditions. All other steps on the selection automatically self cancel.

103. MILK TRANSFER

Step 102/1

STEP 102/1 (ALARM)

Preconditions
1. Step 101/1 Control .And.
2. Step 102/2 Not Active

event; started by transfer selection
monitor; selection cancel

Operates
Checks

1. Level LT501 > L3 — silo S1 level greater than low alarm level
2. Level LT502 < L3 — silo S2 level less than high alarm level

STEP 102/2 (CANCEL)

Step 102/2
The selection is cancelled by the operator, thus acknowledging the level alarm.

Preconditions

1. Step 102/1 Active .And. — monitor; selection initiated
2. (Operator Action .Or. — event; operator interface [102/Cancel]
3. Step 101/1 Not Active) — event; transfer selection cancelled.

Operates

1. Cancel Selection 102 — cancel selection; refer Step 102/1 preconditions

4.4. Software Tags Used

4.4. 1 Flags

Step 101/0	Start
Step 101/1	Initiate
Step 101/2	Open Route
Step 101/3	Start Transfer
Step 101/4	Running
Step 101/5	(Shutdown) Initiate
Step 101/6	(Shutdown) Open Route
Step 101/7	(Shutdown) Start Transfer
Step 101/8	(Shutdown) Running
Step 101/9	Cancel
Step 102/1	Alarm
Step 102/2	Cancel
Step 201/1	CIP of silo S1
Step 202/1	CIP of silo S2
Step 203/1	CIP of transfer line

4.4. 2 Timers and Analogue Switch points

The following (excluding timers) are all required to be able to be modified by the operator external to the PLC (through the operator interface)

Timer 101/0	20 secs	fault display timer on selection start
Timer 101/3	15 secs	initiates shutdown
LT501 L3	10%	low level alarm
LT501 L4	3%	low level cancel
LT502 L1	95%	high level alarm
LT502 L2	85%	high level cancel
FC503 OP1	Zero	close control valve
FC503 OP2	50%	fixed valve position on shutdown
FC503 SP1	~50 m^3/hr	automatic flow setpoint
FC503 F1	~5 m^3/hr	low flow switchpoint
FC503 F2	~10 m^3	flow totaliser preset switch point

SECTION 5 : REPORTING REQUIREMENTS

The following information is required to be saved to disk, with the ability to be able to be retrieved by the operator at a later date.

Production:

Time Started	Time Step 101/1 activated
Time Stopped	Time Step 101/1 deactivated
Production Volume	Totalise quantity of milk transferred (using FT503), during the production time.
Time on fault	Time Step 101/1 active without Step 101/4 active

Shutdown:

Time Started	Time Step 101/5 activated
Time Stopped	Time Step 101/1 deactivated
Production Volume	Totalise quantity of milk transferred (using FT503), during the shutdown time.
Time on fault	Time Step 101/5 active without Step 101/8 active

SECTION 6 : OPERATOR INTERFACE REQUIREMENTS

Selection Screen

1. Include simplified mimic (excluding valves) incorporating
    a. Silos and Silo levels
    b. Active Route eg dynamic lines to show
        transfer line on production
        transfer line on CIP
        silo S1 to process
        silo S1 to CIP return
        silo S2 from process
        silo S2 to CIP return
        water to transfer line
    c. pump to indicate if active
2. Include Operator Entry commands for Selections 101 and 102
3. Include selection status for selections 101 and 102
4. Include fault message displays if any for selections 101 and 102
5. Include page links to Mimic screen; Trend screen

Mimic Screen

1. Include full P&ID drawing showing each real I/O item
2. Valves to indicate Energ/Deenerg Status
3. Valve symbols to indicate valve position (Open/Close)
4. Pump symbols to indicate Energ/Deenerg Status
5. Valves and Pump Symbols to flash if on fault (ie if the position feedback does not agree with the output)
6. Silo Levels to be shown in bar graph form; and digitally
7. Analogue Controller to indicate setpoint, process valve, controller output, and auto/manual status.
8. Auto/Manual status should be able to be modified (toggled)
9. Output valve should be able to be modified if controller in Manual
10. Setpoint should be able to be modified if controller in Automatic
11. Gate switches to indicate Energ/deener status
12. Indication Lamp to indicate Energ/Deenerg status
13. Include page links to Selection screen; trend screen

Appendix B Condensed Interview Transcripts

Legend:
EM is the engineering manager at Anchor.
GM/PLC and EM/PLC refer to the general manager and the engineering managers (two individuals) of the PLC vendor.
OE is the plant operations engineer.
PE and PE2 refer to the two process engineers.
SE and SE2 refer to two PLC software engineers.
SM and SM2 refer to the two managers of software development.

B.1 Production

(Note: Comments 1.1 through 1.11 were initiated by the respondents following the introduction of the software but prior to the interview.)

1.1 That's [simulation] bringing the testing right up to the front end, isn't it, rather than we test at the code end. (GM/PLC)

1.2 It [simulation] would cut down a lot of this design, redesign, redesign that goes on with FDs. (GM/PLC)

1.3 It is easy to get the wrong preconditions, or the wrong valve number, or leave something out all together...the wrong valve numbers perhaps, or route checks, or operations, and maybe even leave some things out, forgot to interlock this operation with that operation... (EM/PLC)

1.4 ...if you could actually test the FDs before anyone started actually applying them you'd probably take out...I think you could say from what these guys are saying maybe up to 50% of the faults could be removed at the FD stage...that would be a really lousy FD...And it could be just a few percent...It depends on the source of the FDs...(GM/PLC).

1.5 I think that the way that this [CASE tool] is going through with creating some formats and standards and being able to check the interrelationships...would go a huge way of being able to up front get a lot of this hassle out...at the end of the day what it should mean is when you go on site you should have absolutely very few chances of being wrong...this would put another layer in front of it. (GM/PLC)

1.6 You can generate some sort of simulation regime too, based on what is contained in your FD. (EM/PLC)

1.7 [Do you think they would have time to do the simulation?] It depends on who your talking to...you're talking about sites that have a shutdown window of 6 to 8 weeks or have a shut down window of 3 to 4 months. And so their ability to generate excellent specification is determined by the amount of time those engineers have to focus on a particular project...in the dairy industry it is highly unlikely that they'll have more than, in a general sense, more than weeks of notice that they've got budget approval to do a job. (GM/PLC)

1.8 [Could you comment a bit about organisational change...10 years ago Anchor had its own designers and programmers and everything...] ...What has happened in the last 10 or 15 years is not only organisational changes in dairy companies, but throughout NZ of course has been dramatic. A good example, you know, [is] pulp and paper industry. You can take Kinley...[had] 120 engineers of which 40 were virtually apprentices and the other 80 were actually, probably 40 of those were maintenance and 40 were design...now they have 6...so the real issue of whether you can get those

6 people to do this doesn't happen. What they would do is they would go out to a consulting group to do this for them. (GM/PLC)
1.9 [Would something like this (FDA) be a nice to have, or a ho-hum, or a revolutionary ... tool for communication among those organisations?] I think it's more than nice to have I think from our point of view, if someone was running a program like this and an FD arrived and you knew it had a 99% chance of working, that would be revolutionary, quite honestly, if they could apply it...Because it doesn't happen today. And that cost to us is...we tie up engineers for significantly longer than we would want to. And we don't get through projects as quickly...So from our point of view, it lowers the risk, it lowers the cost of the project, it lowers the number of man hours we would spend on a project ...I've never heard of a person who has yet been able to say they could give an FD that was tested...It's a major hassle in the industry getting FDs that are incomplete...But your question as to whether the APVs, the Tetra Packs, the Morrows, the all other OEMs out there could do this, present a fully tested set of FDs, it would depend on whether they felt there was a commercial advantage to doing that. And I think they'd be nuts if they didn't see a commercial advantage to be able to put a spec out that was a workable spec...So if they could do, make the production of an FD quicker and more standardised and tested I would think that the process OEM uses would be the key people who would want to use that and we would want to be the recipient of it. CSE, and ourselves...would just love to be the recipient of an FD that was standardised and tested. That would take 50% of our worries out of a job, wouldn't it? The risk would be right down...The time value...we would just chop 30-30% of the hours out of a job. We're in the business of selling hardware. (GM/PLC)
1.10 And so anything that takes all this rubbish out of the front end is of great value to us. ...I see the process OEMs wanting to produce good specs that work. I mean if you take some of the drier jobs around NZ that didn't work because the specification was not correct, it would have cost those companies millions in not being able to put out a proper spec...but I'm just saying that sort of thing has got to be commercially very very good to be able to put these things out in the first run, get it 99% right...but a lot of the things we've see here this morning, I can see a commercial value in a lot of those things today without going any further... (GM/PLC)
1.11 I think... the interesting thing is up, even what you've shown us so far shows a heck of a lot of promise, there is a lot things in there that you could apply very much to the standardisation of the way that we develop data bases, the way we develop...actually identifies a lot of things where there's a lot of variables at the moment already...the CASE tool approach I think that already has a lot of power just up to that stage a lot of things could come out of that that we would find very interesting, don't you...(GM/PLC) ...Not just from FDs, generating data base for a SCADA package...(EM/PLC).
1.12 [Do you see productivity or effort changing for other people in the organisation but maybe not directly involved with FDs and PLC code. Do you see an impact on others in the organisation?] Take less time to engineer things, there'd be less engineers....(E2/PLC).
1.13 [Do an improvement in productivity or quality using the tools as well as the standard?] Yes. [Do you see an improvement in productivity?] Yes (EM/PLC)
1.14 [Do you think the tools would lead to an improvement in quality? Can you think of a measure of quality?] Well if you get your simulation working full I think that's going to be influence on the quality. Once you know the FDs correct, then

just...otherwise you're looking for mistakes. Otherwise you can produce quality on a word processor with the nice fonts and things... (EM/PLC). ...it would always be the same format, when they went to edit it....[OK, it's probably easier to use than the word processor and it's always going to be a consistent format, using the tools...] Yes
1.15 [The report capability [would] the report capability could contribute to quality as well as the simulator?] Oh yes. (EM/PLC)
1.16 [How about down the line in testing and maintenance. I think we've talked about this, the higher quality FD you said would translate into maybe 50% less testing effort] Absolutely. Not only that, from a commercial point view it would create a successful project and the difference between a successful project and a failure is really making money out of it or not. The situation there is if you get it up right in the front end of the project you can almost inevitably get the other end of the project right. And that's where you make your money...so from a commercial point of view it would be very important wouldn't it? (GM/PLC) ...If you get the front end of the project right you can generally track the whole project and keep it on track, and if you keep it on track from a commercial point of view you're not going to loose money on it.
1.17 [How would that compare to current methods, would the tools make it easier to reuse FDs, or more difficult, or about the same?] Depends how tightly you're tied into all those valve numbers, (EM/PLC)... how easy it would be to change all the I/O assignments to the next job....They could do that, I mean Anchor's got the ability to definitely lay out their systems exactly the same from one factory to another. They'd be the only people in NZ that do. (GM/PLC)...So we're saying that potentially the tools will make reuse of FDs easier, if we have the right power of search/replace and things like that? (GM/PLC) Correct, yes (EM/PLC)
1.18 [Some people have told me that maybe there's 20-30% of the code in the PLC that's never written in an FD] Well, that would be right. (EM/PLC)...You could call that making things work (GM/PLC)...Where in the FD does it say you need 10 seconds for a valve to move into position? (EM/PLC)...Checking things, I've had arguments with people about that for years so...
1.19 [I think we talked a lot about the FD simulation and reports and cross references, and I was asking people how they though having those capabilities, like the cross reference, simulation, how that would impact their effort in producing FDs.] Simulation (animation) would be great, wouldn't it, that's the key (GM/PLC). Well that's what really makes this system here. (EM/PLC) [You think the simulation is a key characteristic] yes (EM/PLC)...Even if it went no further than that, I mean even if it went to a full FD that was able to be customised and could be simulated that would really hit one of the key problems in all industries right there. And we talked about before, that would reduce the faults in the system by probably up to 50%. (GM/PLC) [Say short of the code generation, right?] Yes. Even if you stopped it there, then that commercially would be an exceptionally good product because that's where all our risk is at the moment. If we don't get it right at that point we don't get it right anywhere, do we? And it steamrolls, it snowballs into a huge problem that gets progressively more and more risky as you go through the project. If you can get it right there, then you, I don't think there's ever been a project where we've had good FDs, done on good quality FDs, that we haven't succeeded. The ones that we've had real problems with is where they change an enormous amount or there's a lot of errors in it and those errors keep on getting spit out of the system the whole time. And we end up re engineering things by another 30% within the time frames we're allowed.

And that just creates so much margin for error that you can end up with a disastrous project. So I think that's key, the other key is to somehow get people to use it. To present you with an FD that has been quality checked in some form. Or for us in our position, if we wanted to maintain creativity, for us to develop and FD that's been quality checked. Because that maintains our ability to have come creativity in the making of an FD...if the start of the project was a quality FD I think it would be excellent (GM/PLC).
1.20 [Maybe it's turning out that, you know, the payoff is in the front end tool...] My feeling it is. (GM/PLC) [Maybe the code generator just becomes the data table generator or the SCADA message generator, and those are code related things, maybe it doesn't make ladder logic, you know, the logic, but maybe it can make some of these other things....] Yes, because ..messages can be a quite huge list in itself, isn't it. (EM/PLC)
1.21 [Given a choice would you prefer to use the new methods or the current method.] If I was in Anchor or CSE I'd probably say yes ... but we talked about this before, those guys will love it ... to varying degrees. ...if we looked at the front end of the project the same way and said, hey, what we want to do is have a standardised method of doing this, not that it's going to generate standard code, but we want a method of being able to generate an FD, whether it's SFC type of display or whatever. Surely this approach has got a hell of a lot going for it...(GM/PLC)

1.22 [Compared to current methods, do you think you'd be more productive using these tools, less, or about the same as your current methods?] Once you got past the learning phase, and just got into it, you get into a very structured sort of approach I would imagine, which is what we are trying to achieve at the moment, so I think it would be easier because like messaging and stuff like that comes, you are doing it automatically as you go, having the I/O database sort of if you use it gets put in the table, and if it doesn't exist you get told about it. You get that now to a degree when you go to code it, but this here is going to pick that up earlier. (SM)
1.23 [How about the testing phase. Do you see an impact on the testing phase if you had these tools?] Yes, I think if you get your simulator going there is certainly potential there. [OK, so you do see some benefit in the simulation?] Certainly, I think the problem you have there is how far you go. It's like a Pandora's box, the potential is mind-boggling. [So the simulation capability should reduce the effort in the testing phase?] Sure. (SM)
1.24 [...a bit about using the simulation and actually using the cross references, I think you touched on that...You see benefits in those?] I certainly do, yes, simulation for sure, the potential there is quite good. Especially from and end, somebody that's going to, maybe if you, say didn't get to the code generation straight away just being able to simulate the FD, for what we're going to receive will have some improvement there. Use of reports, I see immediate uses there straight away, I mean just the cross references and the summary of, the cross references in particular. And even the summaries of how you were able to extract certain bits, just bring the written description out of the selection, removing all the checks and stuff, I see potential there too. [And you see that in all the phases?] Yes, I think so. [And the benefit there is basically more information, less time to do the work?] Yes.
1.25 [How about maintenance, if we had these tools in the maintenance phase?] Oh, yes, I think there's going to be spin-offs in every facet of it, the I/O data base and the

messaging files, and the actual FD itself, yes there is benefits there, especially when people are used to using the tools. (SM)
1.26 [Do you think the benefits are more in quality improvements or productivity?] Both. [So productivity, would you say productivity is in maintaining the FD or the code or the...?] Both, by having, by being able to quickly update the FD then keeping the code in shape is going to be a lot easier as well because the guy doing that's maintaining the PLC, the code, is got something to reference. A lot of the problem at the moment is a guy does down here what he thinks should be done and then it gets updated back here, if it's easy to do both then it'll be driven from the FD. (SM)
1.27 [If...at Waitoa, if they had Word Perfect up there and they could make the change in the FD with word processor, how would you compare that to having these tools?] I think these tools would be better because it's more encompassing I think. At the moment I think word processor there and making a change, for example those summary lists of where a valve is used, that would make maintaining a hell of a lot easier. If you want to make a change you know what's affected. I mean with Word processor you can search, which is good, but if you can just generate a little report on V101, you've got it all there and then in fact you could just sit down with the FD as well, as I've got time I can look through and check it, whereas with word processor you've got to search and find it and step back and see. Electronically I think people working off a screen, operators and that looking at an FD I think they are going to have a little bit of difficulty in word processor trying to understand, maybe it's me, I find sometimes I feel it's easier to have something written and seeing, ...you've got a better feel for how ... yes, I think the CASE tool will certainly be a better step ahead of using word processor. And you got the I/O data base and that are being managed in their own little modules as well which is certainly, at the moment they don't really have a, they may have a list somewhere...It's more complete. (SM)
1.28 [How about reuse of FDs, I showed you how we have copy/paste and replace. Compared to current methods, do you see any advantages or drawbacks to using the CASE tools in terms of reuse?] I'm not sure about that one. I suppose it's, it was easy to do, it's probably easier to do then what it would be if your copying a complete selection or something like that, it was certainly easy to go and...as you said you could have another one over here totally unrelated and you could, it's probably easier to do than in word processor... I think there's a .... maybe harder in some and easier in others(SM)
1.29 [Would the code generation capability be effective, if it generated the code in the FD?] I think so, yes. When I stand back and look at what we are doing, we're doing a lot of very repetitive work. [How much of it is repetitive?] That's a good question. Checks are different on each step but you are doing the same think so every step, every check, every operate you are doing a similar function. You could say like 80% of it is repetitive, other than that you have checks and you are changing what the check is. Presently in a process selection file, there is the standard stuff at the top of the file to do with displaying stuff on the screen and all that sort of stuff that all repeated and fault pages and process overview pages are repeated in each selection. Within each step you have checks and operates. So I think there is probably 80% is repetitive type. [That code you were just talking about, the code at the top and the code at the bottom, that code's not in the FD is it?] No. [So that wouldn't come out automatically.] OK, Still, I suppose you could have, well, the FD could still contain that couldn't it. You could actually, maybe at the beginning you could say all the selections have to be, see them on the process overview, all faults are displayed on the

fault overview page and as a matter of course, when you are taking, as you were saying, you have a standard template and the bits in the middle change. Yes, it is not in there. You're right. At the moment it's not in the FD at all. It comes by default. [It comes from the experience of the implementers?] Yes, that's right. And that would certainly be, picking that stuff up takes a lot of work out of actual, I mean all you are doing is you cut and paste in the PLC, cut and paste, but still that action of cutting and pasting is not just cutting and pasting, its search and replace and if you could generate that from the source you know that you've got a standard set of symbols for that as well, and you know that you've got to display the name, the step, and the fault and just the middle, the 101 out of the selection name changes, you could generate that probably, it's standard each one, stamping that stuff out....[Do you have an idea of what percent of that stuff that's not in the FD?] That's a tricky question, I suppose it's, maybe, the top stuff and the bottom stuff is probably 10 - 15%. It's hard because it's in each selection, sometimes there's more standard, generic type of stuff than sitting in there than there are rungs for the actual control logic and other times you might have 50 rungs. There's probably 10 rungs worth of ladder in each selection that is standard in every selection. As an average probably 10-20%. That would be an interesting sort of exercise just to have a look at. Because there is quite an overhead in doing that stuff, and a lot of the data base work is related to that stuff as well because you have name, status, step, and then fault displays and process overview pages which you've got to generate as well. [So, if the CASE tool could generate all the code in the FD, that's still not all the code in the PLC?] Correct. [You're saying 10-20% of the code isn't even there.] Yes, I think there's 10-20 that is standard in every selection, then there's going to be probably another 10% that's going to be the customising, the hard, the bits that are outside the normal bounds. [Out of curiosity, could some of it be in the FD or built into the code generator?] Yes. I think probably the 10-20% that we initially first talked about could be generated. I'm sure you could either by mentioning it within the FD or automatically generating. Actually you would do it from the FD because it changes with each selection, you know, like the tagging changes. From what I've seen wouldn't be too much of a hassle. It comes down to what you see on the MMI. Maybe there's going to have to be another section within the selection: Do you want process overview and you want fault overview. Has 5 rungs associated with it. [The other 10%, the customised stuff, is that describable in the FD?] I'm just trying to think of what I'm really referring to there. It's sort of getting down to analogue type stuff, it gets, I'm not quite sure what to say there. (SM)

1.30 [One of the theories that I'm basing my thing on is that CASE tools developed specifically for an industry will be adopted. There's been studies showing that people are not adopting these generic CASE tools and some people have said it's just because they don't fit. They don't fit, like the right language, they don't use the right language or whatever.] Yes, right. [And they call that the domain. Like this is specific to the domain of dairy manufacturing. ...The domain is process control in the dairy industry. That's the domain the FD covers.] Right. [But maybe there's other domains in that PLC. Like...maybe reporting. There's domains that aren't describable as control.] Yes, that's true actually if you put it that way. You've got, I suppose there's 10 files in there that are doing controlling the process overview page on the MMI and there's one that does valve fault page on the MMI that sort of stuff that certainly come as describe in the FD at all. Generic type code that you want in your system is like system stuff rather than process selection specific. If that's what you're sort of getting

at. [Yes, that's what I'm getting at.] Where you have, you've got, there's probably 10 or 15 areas of code that are dealing with process overview, controls, the queuing and that and finding the faults, the valve faults, drive faults, and isolated overview pages, what else, there's 3 or 4 other things, fault control in the PLC, you know, what happens when you go from program to run mode and all this sort of stuff, if you have a major fault, then what do you do, if it's recoverable or all that sort if stuff certainly isn't described in the FD at all. (SM)
1.31 [So there are some limitations and some potential to generate some of that code.] Yes, absolutely. (SM)
1.32 [Given a choice do you prefer the new or the current methods?] Personally, I'd go for the new method. And probably it's easier for me to have that decision because I don't presently write FDs...And I can certainly say that it would be easier for me to produce an FD that some other sucker could understand and implement using your tool than it would be for me to get in Word Perfect and even edit and existing. I think it is...probably even...yes, I think I would certainly find it easier. The end result, I would have a lot more confidence in what coming out the end was going to be useable than what I would if I was going, you know, I have actually edited on FD to produce a copy of it and it is quite a tedious task...I think there's plenty of potential. ...It would be nice, there's things that aren't in the process function descriptions, it would be nice to sit actually down and try to apply what you have built to, like I was talking about, fault routines in the PLC just have a play with that and just see. There's no reason why it wouldn't fit, you're still dealing with tags and bits that have states and stuff like that. And get some output out of it. You know it's difficult for us to play with process stuff, because we haven't really, we've go to do what you've gotta do, get an example, but stuff that's doing fault control in the PLC, and even just trying to apply it to other stuff that's done would be useful. (SM)

1.33 [My first set of questions is about standard FDs, if you think about the current methods that people use...What do you think about productivity, and quality, and maybe other measures as far as creating what you create?] In terms of quality, quality to me is a measurement to conformance. Not whether a product is good or bad. With a tool like that I can't see anything else but being able to deliver a product that is conformant, in terms of structure anyway. I think there should be less mistakes in the document because of the associated checking. In terms of productivity, I'd imagine these guys are pretty quick at writing FDs now, do a lot of cutting and pasting in the word processor, and that obviously is going to affect on quality or consistency of your data. I guess with the cut and paste facilities in there you'd hope to capture that same quality. In terms of productivity I would like to see improvement I don't think it's going to be two fold or three fold, I'd say you're going to get about 20%... (SM2)
1.34 [In creating FDs?] Yes in terms of productivity. What I think it would, I think one of the main things it should enable the guys to do is to focus on the what has to be done rather than the how...the what and how perhaps rather than the trivia of making this bold and that sort of thing...Rather than focusing on the overall look and style of the FD focus on what the actual content is, the meat if you like. (SM2)
1.35 [How about in the other phases of the life cycle. Do you see an impact there?] Oh, undoubtedly. I think the main thing is actually testing and maintenance. I think the productivity will be higher there once you've got the links sorted out properly and putting the links back in the PLC and that's certainly in testing and certainly in maintenance. I guess an FD is like a software program, it takes you 15% of the time

to code and the rest of the life cycle is maintenance, testing, and design. And the FD design is done, the overall structure is done...[Do you see that mainly on the PLC coding side or the FD side?] Oh, 20% productivity gain I'd say on the FD side, but I would say that a 20% gain there in quality is going to be a 40-50% gain in quality of PLC code because errors start to multiply on the way through, so...It's going to have that affect. (SM2)
1.36 Other measures....I would like to think that it would be not necessarily fun, but more enjoyable to use for the FD writers. [The standard or the tools?] The tools. The tools are more enjoyable to use. (SM2)
1.37 [Do you see impact in other areas, getting down to this question, on productivity and effort?] Well, like I was saying before about that's the gut of the plant user manual [referring to the summary reports] as a by product. How about things like this valve is faulted, what is it going to affect...it's one thing to say VP404 is faulting, well that's good, and you might see a selection stopped because of it, but what other selections can't I run, what other impact is it going to have in the plant...now a good operator would know that but you take like those new operators at Waitoa plant they wouldn't have a clue...it's a by product... (SM2)
1.38 [Comment on reuse of FDs] Reuse of FDs, that's an interesting one. You open up all sorts of issues then. [Such as...] Lots and lots of issues. Well in the FD you've got how the process runs. So you've got issues of copy right, all that sort of stuff. [Do the CASE programs have any bearing on that?] The fact it make it easier...And how would you prove that it was an original FD, because the structure looks the same. [So there's an issue of...] Ethics.... think you're going to need some system which encodes the FD data so that Anchor Products FDs can only be read by Anchor Products FD animators or assistants. The last thing you want to do, that's how the process runs, is for companies to find out how a process runs at another plant. [how does that compare to current methods, could this hard copy by circulated?] Yes, and it is ... Through vendors it is... [Would these tools change that?] Well you need some protection, or it will make it worse unless you have some protection. [How would it make it worse?] So easy, too easy, take a disk...if someone is copying a 400 page manual it's a bit different to copying three files... You get the situation is you've got a vendor that makes a piece of plant run at one company and they tend for another job at another company and it's basically the same plant...who owns the FDs? ...I'm not saying I know the answer... (SM2).
1.39 [Do you see Anchor building some component FDs that are reused?] Yes... Anchor would take the CIP template FD and modify it.... Process engineer X has got his own ideas and doesn't want to listen to anyone else's ideas and the only way that Anchor's going to get what it want is if they do that. [So Anchor would say here's some templates....] Here's the standard template CIP FD and code, use it. [Do you see the tools contributing to that process?] Oh yes. [Right now they could give them a word processor file...] Yes, but they just haven't done it. [Would they do it with these tools?] I think it would make it so easy that they would do it, yes. (SM2)
1.40 [Do you think it fits the need...Do you think the tools fit the need?] Yes, The first four applications, yes... definitely. Code generator, well have to work in that. But yes, I think it does. You know, even things like the plant I/O data base. The only think I'd like to see is it ends up bulking onto a real data base at the end....Do ODBC calls or whatever... Yes, I think it does, it's not there yet but it's probably 60-70% but yes, it's getting there, it did come a long way. (SM2)

1.41 [I've got the same questions about the tools themselves. Thinking about the current tools, how would you characterise your effort to, let's say, create products, would you see an increase in productivity with the tools...] Those things don't really apply to me personally but yes, well, it's one of the driving forces I guess, I'd like to say yes to all of those...[using the tools themselves, do you see a benefit to your company?] Yes, definitely. [In what ways?] In terms of the input to the PLC guys...just the quality of the information and the cross referencing, stuff like that. (SM2)

1.42 [Comment on generation of PLC code, data table and messages] Bet you have a few sceptics there....There's a few people saying that it's actually impossible. [To generate the code?] I don't think it is impossible at all. But I mean, the PLC code, I think you should, I can't see any reason why it shouldn't be generated. [... but then what? Like first of all if I generate it... And then it has to get loaded into the A-B programming environment] Well I actually think there's an issue about, you know how you've got a normal compiler and you've got about three layers basically, you know,....I actually think there's about three of those actual compilers in the middle of taking that ASCII, I would actually go as far as you generate to an intermediary standard PLC code and then have a cross translator. [To other various kinds of targets?] Yes. (SM2)

1.43 [But then when we get to the testing phase, are programmers going to be trying to read that [generated] code and modify it?] Yes. [I mean the ideal is no, there're going to go back to the FD] Eliminate the programmers... [But for now anyway, if I'm generating this intermediate code... I asked some of the programmers about their effort in testing and maintenance and they said well, I'm going to have to read that code, my productivity is going to go down because I'll have to learn to read that ladder code that I didn't write.] But that's a lot of crap anyway. That's all bull shit. [Why?] Because there're doing that now. [Are they?] Yes they are. Graham Nobel has commissioned some of Doug's code and, but that's only on one project....Hang on, these guys move from factory to factory. That's an argument but you know, you're doing it now guys. And at least this [CASE tool] is just going too end up being like another programmer, but it happens to be when this programmer code, he code the same way, every time. [How about comments in the code. If the automatic code generator is generating code. One of the comments that a programmer brought up is he said well, we've got comments in there that say this rung, you know, why this rung is here, or this rung refers back to this rung or something, or comments that helped him track something down I guess was his point. He said, well if it came out of this code generator it wouldn't have those things.] Wouldn't it? [It can say these are step 1, it can even pull the comment out of the precondition and stick it in there.] Exactly. What's the, if you had a look at their code that's the sort of things they put in. [So you don't buy those arguments?] No, I don't. Well, my reason is there are 300 PLC programmer code guys in there now and they do go from factory to factory. What's 301 going to do to 300? (SM2)

1.44 [Getting back to this potential and limitation, what...we talked about this a little again, is what code can I generate from the FDs that I have?] Obviously all the selections. And the background checks. See I think there's a step in the middle, well not in the middle but off to the side, that you perhaps haven't got in there yet and you've sort of touched on it in here, to me you've got your code generator, to me it needs like a PLC design process. Someone like Gary goes in there and says, OK, background valve checks are all going to be in this file...This is my hierarchy, it's

called 3 times a second, OK, all the analog code you generate, please put it in file 7 because that's the standard, like an overall global design...it's configuring the code generator...Not necessarily configure....I'd actually call it code design, as opposed to code generation, I'd actually call it design...Because Gary does design, where things go, and designs what the messaging files are and designs...see thing is, you've got an FD half that code goes in one PLC, half goes in another. (SM2)
1.45 [I didn't know that. So all the code from one FD doesn't necessarily go into a single PLC?] Oh no. In fact the digitals are probably held in the 250s and all the analogues in the 260s...that's to handle the process...PID loops and that.... when I'm talking analogues, I'm talking PID control loops, like raising the cook temperature of the drier, the drier temperature, that sort of thing. [And what kind of information goes between the two PLCs?] All the values...actually a sort of message between the two...that actually might be a little bit wrong what I'm saying there in that you can possibly treat that 60 just as a remote controller, you see what I mean...It's a special PID device that does that job and that's how it does it and one step says controller A to this value put set point and that's all handled by a separate, like a super PLC macro kind of thing... just bear in mind that not everything is done in the one PLC basically. (SM2)
1.46 [You do see the potential for generating...] Full code. [100%?] Yes, why not. I think the area's real grey and woolly around the edges and murky because we haven't had these standards and there're different people that code these in different ways. And so you've got all these people that saying it can't be done because we're all different... The other thing is that maybe the effort required to do that analog is not worth it, it's not actually worth it ...The productivity gains are not there, the productivity gains are up in the top PLC...In terms of what you want to generate and what's possible to generate I think are different...I'd still say it's possible 100%. People code it, people code from this FD....
1.47 [What about the problems down there in the testing and maintenance where I was talking about people modifying the ladder and bunging in the ladder changes in online?] Hmmm, yes, that's I see two distinct uses, I don't see necessarily anything in between. You either, you write the FD and you print paper and that's condition A or you can write the FD and generate code and that's condition B. [Could you have condition C where... the code is the data table and the message data base and that stuff but not the logic part?] I think the two are all linked together. What happens if you delete a valve? What happens to your logic for that valve? Well maybe you could, the person would have to go through by had with the cross references and delete them. You're loosing that inter-operability between data tables and PLC code and text messages, aren't you, you're loosing the linkages there. (SM2)
1.48 [OK...I guess we talked about this effort to analyse alternatives in FDs, with the cross references and simulation...do you see a big pay-off with those?] I'd say so. Yes. [In productivity and quality?] Yes.
1.49 [Given a choice do you prefer the new or the current methods?] ...no choice really....new method, I can't see that, the current one's too unstructured, too easy to not adhere to anything...do easy to make mistakes. (SM2)

1.50 [I have some of the same questions about standards relating to the CASE tools themselves. And you've kind of mentioned that actually a little bit in you're, previously. I was interested in the effort to create FDs or code using these tools. If people thought their productivity, quality of FDs or code would increase, decrease,

about the same. And, looking at the different phases.] Are we talking here about if this machine was to generate code? [Yes.] Well, then, yes, productivity should sky rocket... As far as having no need to have a coder to do it. And obviously the quality would improve as a result of that. (EM)
1.51 [That would be the creation, what do you think about testing or maintenance?] The testing phase again is that scope of using that machine to generate that simulation code. If you don't go for code generation for what gets installed in the PLC, go for code generation that gets installed in some simulator module, there'd be a lot of savings there. Yes. (EM)
1.52 On the maintenance side, though, if you take the thing to its full scope where it does code generate for what's down loaded into the PLC, for the sort of fault finding issues, the maintenance person won't be able to look in and understand, what's going in, he's going to have no documentation, because your program is generating op codes or whatever, to view some graphical ladder representation of that program, I'm sure you can view that on the A-B programming terminal but to have layered on top of that all the comments and things which when a person's writing ladder they annotate it with all that sort of comments, you're going to miss, not going to have that... So in the fault finding stage it's going to be harder for the maintenance people if you go to the full code generation... You see, at the moment they'll look through wades of paper with the ladder listing with all sorts of English comments written explaining what the program's doing so they can sort of follow it. But to have all the ladder there without the comments is very difficult to follow... It's that sort of comment like I adding 1 here to this offset because of this little trick we did... back up the program...that sort of programmer's comment wouldn't...(EM)
1.53 [Any other comments you might be able to make?] The reports that you've got there I think would be very useful, the cross reference reports, which actions use which items and all that sort of thing... [... the summaries and the cross references, do you think that would be a productivity aid in the creation stage? Creation of the FD or the code?] Yes, and the maintenance as well... [That would give you the ability to know where an item was used or if it was used at all....] Yes. (EM)
1.54 [I'm asking people if they prefer the new methods or the current methods?] Obviously the new method would be practical I think. The structure would take you that step closer to code generation. Code generation is going to come eventually, just how long it takes, it's a step in the right direction. If the overall objective is code generation then FD structuring is just a necessary step to achieve an end. [How about if the overall objective is just standardised FDs?] Well, that's a worthy objective as well.
1.55 [Comment on using a generic development environment such as TI's as compared to this system.] This is evolved from the paper system and that paper system has evolved from the constraints of the environment that it's used in, the PLC programming language and the PLC structure and the PLC code and everything had to be a certain way so the FDs were written in a certain way to enable the programmers to do that quite easily. If you never have to worry about what's inside the PLC then there are methods graphically to represent a process better than just text and tables and words. To have this system where you somehow graphically not just represent the logic but graphically represent the physical interconnection of the processing equipment...the next step onward as well have it generate the code... (EM)
1.56 I think that scope there, that simulation, code generation will save a lot of time and it's something that is achievable because it's, especially doing something that's

within your own constraints sort of thing, when you're controlling a factory and it's actually going to go into a real PLC to control a factory it's always, well you know, if it's Tuesday afternoon and it's after 5:00 and the guys had his cup of tea then you do it slightly different in the end then try and build all that sort of contingency logic into a code generator that's really going to control a factory and it's hard work but for something like a simulation it's more achievable. [so you're saying, make sure I understand that, maybe the best approach for now is to not try to generate the actual ladder logic code, but to generate the code to drive the simulator.] Yes. [But I could also generate some code that helps in the programming of the PLC like the data table or the message data base, especially...] Yes. [...like I can pull out all those fault messages and I can pull out, there's other text messages they put in, like they say display preselected when they're in this step, or initiated, I can pull those out automatically and generate tables like that and they could maybe be exported in just a little ASCII file that could be imported into whatever...] Yes, whatever... [And that could kind of fit along the lines of maybe the simulator. It's some code that, you can generate it once and it goes somewhere else and it's not this long term maintenance kind of thing.] Yes, that's right. (EM)

1.57 [I'm asking some people how they perceive using these tools, the effort to reuse FDs as compared to their current methods. If they thought the new system was easier or more difficult to reuse FDs than their previous system.] I would think it should be a lot easier, because the whole thing is more structured, it's not just one great big text file, because it's more structured it should be much easier for reuse. [Because it knows about selections or steps?] Yes. (EM)

1.58 [I want to ask some of the questions... relating to what you see here in the tools themselves.... about your effort to create your code, test your code, and maintain your code if you had these tools. You'd have the standard but you'd be using these programs to use the standard. How do you think your productivity might be affected in creating your code if you had these capabilities implemented in this way?] Should improve. Yes, for sure. [Decrease time to code?] Yes. [Improve quality?] Improve quality so far as it would improve consistency. It's all dependent on what you are working with. [On the FD?] Yes, on the tools. I mean the first time around mightn't be brilliant but, looks pretty good, the next time around improvements and enhancements (SE).

1.59 [How about the testing and maintenance phase. Do you see do you see benefits or drawbacks?] I think the drawbacks would be the testing, the fact that you are go in and you go well, at the moment, change the code there and then, and if the code's generated from the code generator, then it's a matter of spending the time and methodically going back and making sure that you've got it in the right format so then when the code's regenerated after simulation to keep the two consistent that you've done that accurately to keep things in sync then if the code's not been generated also doing a compare, it's hard to do a compare with what ...so that could sort of be a bit hard... [Then you are saying if we went and changed the FD and regenerated the code then you wanted to be sure that what you thought should be in there is in there.] Yes, it's almost you'd need to simulate it again to make sure that everything happens. Although that's just a matter of being methodical. But it is, the process you do. Then when you get to the maintenance phase, it's a lot of work to keep the FDs up to date, but although it's a lot of work but it's a lot of code and a lot of process is dependent on it so its got to be done. (SE)

1.60 [I'm interested in talking to you about the PLC code and what the potential and limitations were for generating code and I know there is other code in the PLC than what's just in an FD. Is that true?] Yes, sure. ... at the moment we'd say, step 2 would say open, step 3 start transfer, step 4 running, maybe from that you could say in that moment I'm in step 4 and a message related to that step or whatever, so although in the FD it doesn't actually say display that but from the fact that you could say that step 4 is the running step and you could inherently put that in. [I could put that in automatically, right?] Yes. [Is there some code that I wouldn't know?] There's also, all the background sort of checks, the fault I would use, the process overview pages, which you wouldn't really want to know about. I mean, to get it into a written format to regenerate it wouldn't really serve any purpose because once its there and once its developed there's no real point in putting it on paper and reload it again. [Do you reuse that code?] Yes, its basically cut it out and put it into the next system. But as far as, within the selection, basically your selection starts off you clear the integers, your fault, your step, all that sort of thing, and then you go through pretty much to the FD, and then there's a fault delay timer which as it's stepping through gives the valves and the main valves time to position before saying where are going to fault this valve so that's also a... maybe another little, which is, would be the same for every selection, so maybe that's another thing which although its not in the FD is another little piece you'd want to put in there. And then also update the process overview page. [That's for the MMI?] Yes, and fault overview pages and things like that but that all can be done in background tasks and things like that. [There's no FD for the background tasks at this point?] No not as such, strictly speaking is all extra sort of thing but it's all there. [... if we tried to make this happen for real, would you say there's another chuck of ladder code over here like background tasks that were written in ladder code to begin with that get combined. So I have some ladder code for the selections, and maybe I have some ladder code which is background tasks] Yes [And those would actually have to be brought together?] Yes, and also, there's also another type, your sequencers and things like that which... that starts to get more complicated and [Is that for your CIP?] Yes, for CIP and also for pasteurisers and things like that. So there's certain things where you may want to put those in individually by hand sort of thing. [Do you think it is feasible to have some automatically generated and some manually code and put together?] Yes. [They would just go into separate files?] Yes, they are not in the same files. (SE)

1.61 [Can you give me a rough percentage how much is the actual from the FD and how much is this other stuff] Probably from the FD is probably 80%. [So 20% of the code in the PLC is not from the FD. There's no FD that describes it.] Yes. Even at the moment, just by a lot of things that are not strictly in the FD, but you can take from the FD, that maybe they'd want to see this, which would probably, that sort of goes into this 20%. [So under current limitations, even if I could generate everything in the FD, there's probably 20% that's no FD?] Yes. [can you think of any other potential or limitations as far as generating code?] No (SE)

1.62 [So you're saying this background code is more for the operator interface and reporting what's going on?] Yes [Tell me if this makes sense, but this FDA lets you specify the control code.] Yes. [It is made for the control code.] Yes. [But there's another application really, and it's reporting to the operator.] Yes. [And so there's really two applications running in there and I'm addressing one.] There's almost three because there's also reporting which is generated back into SERB and so that's more information that is grabbed by the PLC and sent back and also there's this other, what

I was just talking about...[This SERB?] Yes. [Is that the factory messaging protocol that there're trying to use. There trying to send it up to the higher level machines for...] Yes, for whatever...report generation. [And that code, that logic is not in the FD?] It is, that code is in the FD but whether, I mean I guess it could be, it's not actually operational, it's when this happens capture this data and set this flag so the, I can capture. [Yes, so you capture it. There may be some comment in the FD that says I want to capture the start time, stop time.] Yes. [And you just capture it and put it into a register and there's something polling the PLC. Is that right?] Yes, it pulls that out when a flag's been set or whatever. But what they're tending to do now is tend to write a separate reporting FD, and saying, so that's in a separate structure, and they have their own subroutines for what all that happens. So it's almost another off to the side. (SE)

1.63 [This may not be relevant to you, but I was, maybe it is. We have the simulator and we have these reports like the cross reference. And I was going to ask if that helps in producing FDs or understanding the FD, having these capabilities.] I don't think it would, I mean.... Maybe if you've got your cross reference, and a valve is on for some reason and you can't, and you've got your reference, you can go and see why, what are the other things. We actually do a lot of our coding and changing and monitoring use cross references within the PLC on line. [So you've got cross references within your PLC programming environment?] Yes. [But this (CASE tool) is targeted more toward the FD writer. He doesn't have those cross references like you do.] In some ways it would be quite good in the fact that maybe actually yes, come to think of it, in things like when you've got a dynamic piece of plant such as a CIP systems which is pulsing valves and things like that, and ideally you want to check that a valve is in a particular position, when you energise it, it must be energised, but sometimes you can't check for that because during the process it is opening and closing, for some reasons, so what we do then is check that it is at fault, that it is not at its correct position, but with a cross reference, the FD writer can look at it and say well these are the conditions that are operating the valve, so while these conditions are not happening I want the valve to be deenergised because it should be deenergised when nothing else is operating it, so in things like that, yes, you could have an advantage, and that way that would be an improvement, because instead of saying it's a fault, I'm checking that it is deenergised because it should be and you're physically making sure that the valve is in the correct position. [And you're saying a cross ref would help with that?] Yes. [Because...] Because you could find where things are being operated and it would help with checks within other selections... And if we had that then it would... there may be facility for adding to it later on as far maybe other things you want to check or at the moment I've, actually just this morning I was writing down ideas of what we could do, and it was in fact to generate an auto bit so that everything that was running on auto from selections, turned on an auto bit, and the auto bit turned on output, ...that sort of thing... And then while you're looking at the auto bit there's maybe the possibility in the future that you might use it for doing different types of checks.... I mean what I was originally thinking was have the force on there and the actual force on/ force off, auto bit and the valve here but maybe you just have it in a mirror file, the auto over bit here and the valve here and in reality you have this in file 2 and this in file 3 and the same rung would correspond to the same valve so you could jump between the two easily. (SE)

1.64 [So you can see some benefits to analysing the FD...having the cross reference could help the FD writer. You already have these things for yourself.] Which was

automatically generated... [For the FD writer, were you going to say something?] I'd say that the first time through maybe it wouldn't be any use because it's not there, but once he's generated his cross reference and he's going through and checking, back checking, that he's got his cross reference ...(SE)
1.65 [I had one question about reusing FDs. Maybe we've addressed some of that, but that's like the copy/paste and things... You've talked about we could have more power in the replace capability and maybe super replace, wild cards, or the matrix idea. If we had those kinds of ideas do you think these tools would make it easier to reuse FDs or parts of FDs compared to current methods] Yes, well current methods are cut and paste, search/replace, but in some ways there may be a tendency to use FDs that could be improved upon instead of there may be some aspects of while you are copying you don't tend to do that if you don't rewrite it then maybe you would try and improve on it the next time. So there could be a draw back if you take too many short-cuts to generate an FD rather than...[You mean if we make it too powerful it might be tempting to just grab this and put it in and not do the rewrite that maybe you'd have to do with the other methods] Yes. (SE)
1.66 [I was curious how having the standard and the CASE tools might affect sharing FDs maybe with other people in the organisation. Do you think there'd be any change in the use of FDs or sharing FDs?] Sharing as in regards to within Anchor? [Yes, within Anchor probably. Taking an FD from one place and reuse it in another place. Or an FD written by one person, like maybe Murray wrote an FD with the standards and Brian was able to take that and use it.] I guess there'd be a tendency for that to happen. If it was, it would be definitely an advantage if you could get an FD that was already written, financially for a contractor to go and grab that FD so there may be, probably Anchor's attitude would be the price has to come down, you are already given this information... [Right now Brian has copies of all of his old FDs and he can reuse those, but he doesn't have access to Murray's FDs does he?] No but there're not a secure document anyway, you can go in and just borrow an FD with not too much worry it you want to do that. [Yes, but I guess you'd borrow it in a hard copy form and then you'd have to look at it an adapt it.] Yes. [Whereas if we had this standard...] With that there'd probably be this tendency well, it would be just the fact that it's there and it'd want to be used as a standard so the tendency would be everyone just go for it. The standard would probably more easily kept because it tended to go on. (SE)

1.67 [I was thinking the current tools, the word processor I guess and Microsoft access] Yes. MS Access. Tying it all together that's the problem, it's all so fragmented there's so many little applications for what where're doing, various bits, importing an I/O database and creating I/O messages from that then internal messages to be linked together with the I/O to go to the operator interface. That type of thing. Keeping track of that, centralise... [Integrating those disconnected tools into one. You see that as a benefit, making it easier to use?] Yes, definitely. (SE2)
1.68 [I have some questions about the tools themselves... would you see your productivity in creating the PLC code increase, decrease, or be about the same?] Well, hopefully, it better increase, if it is supposed to be generating the code and taking the drudgery out of it. (SE2)
1.69 [How do you think it would affect your productivity in testing... commissioning?] Well, as long as it creates a good PLC database, it depends what it spits out at the end... [What do you think about having to look at code that you didn't

write?] That's no problem as long as you know the approach taken to it, perhaps a system type, a structure chart of some sort... the biggest thing when you look at PLC code for example that you haven't written is you don't know the approach they've taken writing it, you don't know what their philosophy is, how they've actually thought about doing checks and displaying things to the operator. At the moment we know, we know certain bits of data table are being used, and we know the PLC scan, how that's being derived. What we don't know is just ladder logic, at first glimpse it could be doing just about anything. [... so what you're saying is if the code generator produced a known structure, then you don't see that as a big impediment?] Yes. [So you're saying in the testing phase you don't really see a decrease in productivity?] I don't think so, because it's standard once you've done the first one, that might take... [Initial learning curve?] Initial learning curve, but then it should be all the same. As long as it's not spaghetti logic. then we're right. (SE2)
1.70 [Same with maintenance phase? Would the same comments apply to maintenance?] I think so. (SE2)
1.71 [About quality. Do you see and improvement in quality in testing or maintenance, or about the same?] Well, it should be more consistent, I think, once it's been developed. Hopefully the consistency is one thing that sort of leads to a processing plant, consistency leads to quality as far as I'm concerned. (SE2)
1.72 [Any other measures of effort you can think of, or effectiveness, or impact as far as creating, maintaining?] No, just time, centralising the different operations that you do. It's pretty important that it should all be automatic, the PLC database and I/O database. [When you say centralised you mean the same structure?] Handling it, automating it.... [I see what you are saying, you mean centralising them into the FDA instead of this little tool or this little application?] Yes. (SE2)
1.73 [I showed you a little bit about the reuse, the copy/paste... Compared to the current methods, do you see a negative or positive?] I think it's positive, you definitely need it. At the moment we would do the same thing. We write a selection and then copy/paste. And then go through an just change the FDs... [Would you say it's about the same effort to do the copy/paste with the new method compared to the old, or do you see a negative or positive with the new one?] It should be positive I would imagine, because at the moment you'd be working within a selection. You copy a selection then you're working within a selection at the moment you copy something then you might be within a document which has got other selections in it should hopefully be easier to manipulate... Less chance of error... At the moment if you've got a big selection you'd probably copy it out to a file, change it all around, copy it back in. That's what we do at the moment... a lot of selections are really the same, there's no point in having to retype it. That's shows the practicality. What I've seen so far is practical, that sort of thing, which is a good tool. (SE2)
1.74 [Next question is about the potential or limitations of actually generating code. What I was getting at there, and we've already talked about it a little bit is I know there is some code that is not in the FD, and you said that maybe 30% may not be in the FD, it's that overall shell, so that was one of my questions here is what are the limitations of code generation, one limitation is that if I don't have an FD or some description of that overall code I obviously can't generate that. So that's a limitation in the development phase, there's some code that I can't generate. Is that code rewritten for each application or is it reused?] No, hopefully, once its developed it's reused almost without modification. As long as we use the same data format, database structure, we can just get process selections and plug them in... that's the

type of thing that can be changed which means we have to rewrite certain portions of it. But other things like process overview page which displays current active selections and any message or fault, that can be just ported directly... [So it may be 30% of the code but less percent of the effort?] Oh, definitely, yes. It's overhead, once it's developed, it's done. [It's like an overhead and once it's developed, what percent of the effort would you say it is?] Once it's developed, I suppose 10-20%. To be sure that it's working properly. Once it's developed all you need to do then is to update the database ...We have to update the database with the selections that are actually there. But it's only a matter of changing a description and the database is up to date. (SE2)

1.75 [Concerning the simulation capability for doing some what-ifs on the spec and then things like the cross references and summaries, I'm curious whether that would increase or not the effort or decrease or not the effort to analyse alternatives in FDs, could it help in the production of FDs or in the quality?] Oh, I think so. If you can prove that the logic works... I don't think it would replace the [current] simulation. [In] the simulation, they see what they are going to see at the plant. It's not just simulating the logic of the code... What they see in the simulation is what the operators are going to see. That would help you just prove that a piece of logic is feasible, so that you don't keep duplicating errors. It wouldn't be worth it if it didn't work really well. (SE2).

1.76 [How about the summaries and cross references, do you see those as big time savers or maybe not?] Oh, I think there're good. [Do you think they would effort, save time?] Yes. [Maybe help with quality?] With quality, yes. (SE2)

1.77 As a final form of documentation, you can see what a plant is supposed to be doing, not just in the development stage, but plant maintenance. For the operators that you give them a process description, they can get deeper by the level, they can have an overall process description, then a selection description, then a step by step description, they can look at what they want to. It's quite handy to have that not necessarily have that on paper, be nice to be able to be able to access it easily, view it on the screen. [So you see a benefit in having the on line, rather than just the hard copy?] Yes, it's nice to have the hard copy but I'd prefer to have something, I'd like to be able to access it on the screen. And then you can just put another sections of it, hopefully you can do that, sections of it that you are interested in. (SE2)

1.78 [Given a choice would you prefer the new or the current methods?] Why would you want to switch? I'd like to use the new personally. I think the new environment is better. I like the graphic interface windows interface system, I think it is a good productivity tool in itself. You know, with you pick lists and in built checks. It's a good productivity tool. I think something, reports, final hard copies, might need sprucing up a little bit. Make it easier to read, that sort of thing... Perhaps using different fonts... Yes, but that's only a matter of a couple hundred hours, like I say, getting it onto paper. That's another thing. (SE2)

1.79 [Potential an limitations of standards on productivity?] It won't make it [creating FDs] quicker...But...would make the productivity better if it works well at the end of the day it would not necessarily be slower. (PE)

1.80 [PE's first comment on simulation:] If the code is going to be automatically generated then is there much advantage to do it at this stage, because you might as well go ahead and simulate the code and have the operator interface to be able to see

things more. If the coding is done automatically then that becomes unimportant. [compare to next statement, made later]
1.81 [Do you see an increase or decrease in productivity or quality in any of these phases using the CASE tools?] ...I was hedging a bit before probably thinking about the current implementation. There's no reason why productivity shouldn't be the same. I notice a difference going from one tool to another. I've been using one common word processor for a number of years and I've been changing and my productivity has gone down. There's no reason why there should be any difference. There should be some improvement, there's some things that could be done in testing which I don't have the ability to do at all at the moment so anything that can be done in terms of testing must be an improvement over what we're doing at the moment. [You mean the simulation?] Yes the simulation or the cross referencing. That all comes back in the testing too. (PE)
1.82 [Do you think the CASE tools would be more difficult, easier, or more time or less time to reuse FDs as compared to what you do now?] There are one or two replacements like that replacing whole selections or whole groups of selections where what I do at the moment may be a bit quicker, I can think of some other things that you are doing in there that might make things a bit easier. The ability to change one step with another and automatically cross referencing what you've copied. I think that's fairly similar to what you've got. Dwelling on the overall impression that I'd like to see that would assist productivity are the ability to see more things on the screen and what comes out of the print out. I can't think of many things at all that would be less productive and many things would be a plus. (PE)
1.83 [What % of code in the PLC does the FD account for?] Well at the end of the day it should be a very high percentage. There may be things that we don't do at the moment but only any standard background programs that's put into the code there's no reason that an FD could not be written and added to the start of every project. (PE)
1.84 [Impact on productivity or quality of simulation or reports (see also #20 under C.4)] These summaries and cross references and so on must be a big plus. The use of reports to report messages and cross references...[Do you think those would reduce time and improve quality?] Yes, they would. (PE)
1.85 [Given a choice would you prefer the new system or your current methods?] Still at this stage, the current methods because of the tools, I think, the output [format of printed output] tools...Why would I want to switch? I'd want to switch for all the advantages we talked about before: standardisation and keeping things the same and the advantages that are going to be developed in terms of the standardisation. In terms of people understanding what I'm talking about. There's a big plus in working to a standard that we are not doing at the moment. I think the only standard that people are talking about at the moment is the Grafcet type approach. Because it is the only standard and it is being promoted by some developers as the best way of doing things because it is a standard, everything else is not. We are not working to be best system, but we are also not working to a standard, so that's why I'd want to switch. Given a choice I'd want to work to a standard that helps everybody else, that aids communication and things...[So you would want to switch if the tools were adequate. The standard is not the impediment. If the tools had the formatting and display capabilities?] Yes and ease of quickly making changing and things. I can imagine that when people start using that, the same as any software you'd want to make modifications to the thing that you will only see as you use it. But in general, the fact

that you've got a system that is easy to learn has got to be a major plus. I can't see in a lot of other tools, the learning phase is a major task so that's not going to be there. That's great. (PE)
1.86 [Note: See also comment in 2.34]

1.87 [Do you think it would take more or less time to actually create an FD?] It would take less time. (PE2)
1.88 [Do you thought you would be more or less productive at creating your FDs?] Some sites, when I do an FD, I have to insert the I/O actual number. Some such request the I/O location in the FD. [How do you get that? Do they give it to you?] That's the problem. I have to go and get the I/O lists. Bring them home and as I write the FD insert the I/O at the same time. That's laborious, because every time you go and close SB4201, you gotta go and chh, chh, chh find what the hell the I/O is. [Yes, [the tool] has a pick list.] Yes, that's sitting right there... [So do you think there's potential then maybe you'd be more productive with these tools?] Oh, if I could just put V4202 and It'd go with the I/O number right there, it'd be magic, magic. (PE2)
1.89 [How about quality of your FDs. Would they improve, not improve, about the same?] Oh, I am for perfection in those things, so I don't think I can get better than I am at present. [So you would expect your same level of thoroughness?] Yes. (PE2)
1.90 [Do you see any benefit in the testing or maintenance phases with these tools, as far as your concerned?] Well, if you had that tool sitting on site during commissioning it would be magic. At present I mark up, during the commissioning period, we've got this book, as we're going through any errors we mark up and I go home, type them all in. Whereas if that was sitting there right beside the operator interface, we say oh that's wrong, and change it right there it would be excellent. [How about, could you take a laptop PC out with you? Oh, yes, could do, yes. [Would this be better...?] Well, it would be as good. No, it wouldn't be a benefit. Either way you are typing corrections either in there or in there it doesn't really matter. Because the job's just a typing effort to correct things...I think it would be very similar to change a line of word processing as to change one of those. No I don't see much difference in that. (PE2)
1.91 [I showed you a little bit about the copy/paste and the replace. Do you think it would be more effort to reuse FDs or less compared to what you do now?] The same...I use the find/replace with prompt. I never use if off. I've made some terrible errors. (PE2)
1.92 [If I could generate, let's say I could generate the messages associated with this FD, and the data table and the ladder code, what code, what percent of the code is that that's in the PLC...what percent is accounted for in the FD vs. this other code that's in there that's not in an FD?] Well, you're asking the wrong person...But certainly if it's background valve checking where every valve in the factory is checked in four states. If you haven't got the output on you check that you've got the deenergised feedback and you haven't got the energised. If the output is on you check that you've got the energised and haven't got the deenergised. And for every valve in the factory there's massive software, I don't know it takes, but there's a fair bit to it with untold timers and bits and pieces to it. (PE2)
1.93 [I showed you the idea of simulating the FD and I showed you some of the reports like the summaries and cross references. Do you see those as impacting your productivity or quality of your FDs?] The simulation part is, because of the way we

work, unrealistic. We don't get the luxury of time to do that. We pump these things out as fast as we possibly can and while the paper's still hot it's sent to the software guys. [So you don't see yourself sitting at home what if I energise this or that?] That's all done up here when we're writing it down. So I can't see me ever using simulation at that level. Simulation at the PLC level is magic, brilliant, and saves us heaps of time, but just at the moment my backbone tells me at that level is probably of no value. (PE2)

1.94 [How about automatically generating things like cross references, would that help you?] No, I can't see that at the moment, no. I'd love to have a table of contents, that's my bug there. My presentation at the moment, there's those things, the only way I can get this to work is put a dummy line at the top of every selection, and the Word Perfect picks up that text there and that selection number and stuffs it in here. And it's ugly because it doesn't put gaps, logical gaps between selections. That's a curse too, trying to get contents page working so that the thing automatically numbers itself. It's quite messy...One of your reports would do that of course, so I'd certainly use that part of it.

1.95 [Do you see a benefit in some kind of a report that, I guess the ability to make, let's say you populated that real I/O data base with the plant items you were going to use, we could potentially generate reports that say these items never appeared anywhere. Pump 102 is never referenced. Would anything like that be of benefit?] Well, we do that at the PLC level, we do find the odd output that's got no code to it and that alerts us to a problem. And certainly the earlier we pick that up the better. So before you present the FD you do that cross reference and find that oh, I forgot the pump or something...It's hard to say till you've actually used it. (PE2)

1.96 [Given a choice do you prefer the new or current methods?] I prefer the current methods! [Why is that?] Because I am selfish, and I've been using it for 15 years, and I know it backwards and I don't want to change...I'm being human. [...is there any reason you would want to switch?] I'd want to switch fairly quickly if it impacted on my income...Then I'd rapidly change. No, there's benefits to this, there's no question about that. You've done a wonderful job here. I'm highly impressed. It's really good. In the long term its got to be better. I know of software guys in town that have torn their hairs out by having used mine for a couple years and having generic code in the PLC to suit my layout and then have been struck with different FD type of presentation from other people that didn't suit the generic code that was there, and they had to redo it and try to match it up, and all this sort of drama, so the sooner we get it standard it's going to be much better for all software guys, it has to be, it won't affect me much. [Yes, one thought is if there is a standard, as long as the way of writing FDs is a word processor, the word processor is not enforcing you to really follow the standard, whereas something like this forces you a little bit more into the mold and...do you see that has a benefit of the tool, having the tool itself, is that it forces, it kind of imposes the standard, rather than with the word processor it is up to you to follow it. Do you see that as a benefit of the tools?] Yes, it has to be. It's got to be better for everyone in the long term. Present something in a standard way is got to be better than fragmented ways. (PE2)

1.97 [I showed you the idea of simulating an FD, do you think that would have an impact on productivity or quality....this capability of simulating an FD before code is generated.] Certainly when you are writing and FD, when you are out there and you have a contract to build a plant and obviously the simulation, success of your

simulation, the fact that you are trying to get out all of the bugs before hand is directly related to productivity in commissioning, the more problems you can get out during that part of the FD writing or simulation, the better it is for the commissioning engineer who is trying to get that plant running. The less problems you can get out at this point, the much easier it is for that person. Any problem which is taken out at this point, or any problem which is still left in at this point probably magnifies 10 fold... Is magnified 10 fold for the person who is... [How about those other reports like step summaries, selection summaries, or the cross references, do you see those as helping...] It's another tool really to debug your FD. [OK, good. So ...the cross reference should help to debug the FD which makes it much more efficient later on in the process?] Yes. (OE)

1.98 [And maybe this doesn't apply directly to you... I was asking these different people if given a choice would they prefer to use these new tools or would they prefer their current methods.] Yes, a bit hard for me to answer, but I think given what we are trying to achieve here, I would prefer to see a structured standardised FD writing system in place. (OE)

1.99 [I have a question about how these tools might impact his ability to reuse FDs...Do you have any feel for that?] Well, I think that's an certainly an obvious benefit I would think, you would be setting up a library of modules for every part of the process. That's got to be where your big benefits are. Big time saving, I've talked about before, about a separator, you run a separator there are certain things you can only do for a particular machine and if you had this out in a library, it's pretty easy to pull it down for another plant. Pasteuriser is the same...there are certain key things you need to do... Using the whole FD....I suppose that's probably possible within say a single process plant, you might have five or six say tanks for example, and each of those tanks may be feeding six separators, so you've got essentially, I don't know what it is, a combination of source and product from one point to another point, and obviously it's going to be pretty easy to map each of those FDs for that particular part of the process. But to then perhaps go one step further again where you're looking at an entire wet side operation, perhaps that's a little difficult to conceptualise at this point but that may be something we should also be considering where you are actually just keying in your number items of plant and the FDA is intelligent enough to work out all the specific routes or selections you should actually look at. [So it would be having some rules, some knowledge of the plant in there?] Yes. [And you could feed in the particular items, and it could set up routes or something for you?] Yes. (OE)

1.100 [...About reuse, if you had smaller, if you've broken it down to a smaller level, maybe that would be transportable. Does that make sense?] Oh, absolutely. [Besides pasteurisation and separation, are there other?] I'm sure there are....product transfer....CIP, you know there's certainly parts of CIP which are pretty transportable. There's other components of plant out there, of course, homogenisers, scraps of heat exchanger or....when you get into bigger plants like your evaporators or your dryers, perhaps it's a little bit more difficult to transport. (OE)

B.2 Coordination

2.1 ...you can have more than one person writing FDs...(EM/PLC)...[How practical is it to have multiple people?] At APV they have 2 people writing...So-and-so started and so-and-so came out and you can tell the difference...They can split the plant up....
2.2 [How about...controlling changes to the FD...] A lot of people don't even have an FD to look at. The most documentation they would have is some sort of document of ladder print out...yes, it's terrible, so they don't even have an overview of how their processes work. And that in itself, if they could have that, I think it would make it a heck of a lot better for those people working within those industries in their understanding of their processes, because I'm sure they don't understand them. (G/PLC).. So I think it's not only controlling but exposing people to their own processes. I think Anchor has done this very well in the way that they develop these things they do involve the production people in the development of their FDs, in the way the plants going to run, the layouts, and all sorts of other things, the graphics display, and all that and they do that very well, but they'd be one of few people that do it well...An educational process as well...
2.3 [What do you think about the potential limitations of my ultimate model if I had to generate import/export code. I could like do an initial load of some code but then in the testing and maintenance phase I loose this incremental capability....] Unless you're editing at the FD level, it's lost isn't it....I just think it might be useful to turn it into a function chart, because you've got most of the elements there. A function chart like display the intent....well than just words...[Have a Grafcet view of the FD and have a text view of the FD?] Yes (E1/PLC)
2.4 To keep the model functional, you know when you change the preset on a timer you want to change the FD at the same time. (EM/PLC)...Yes. And what Doug is saying you could change it at the FD level but you just wanted that, and you might want the option of going on line...Other than the fact that it could be a tool up to commissioning but not during commissioning. It could still be a tool almost up to commissioning but during commissioning and afterwards it never could be, could it... Because you're doing things when the plant is running and you just can't shut it down. That's the issue really isn't it...beside the ASCII import/export, there must be some protocols for this...
2.5 Well if you had that generating code and then you make changes then you go to the A-B software and lop that file in, down load it...so you do the changes with the FD...You just need to do a history record...change would be lopped in later....(EM/PLC)
2.6 I think the other problem is, I'm not sure that organisationally, which is going back to what you said before, whether it's even totally feasible, that people would always make the change on the FD. I mean who would make the changes on the FDs? (GM/PLC) What I'm getting at is who would make a change on a FD and what areas of the plant would you allowed to make changes on the FD. Since most big chunks of processes are under some sort of process guarantee....It is a very real problem though, because you can look at a lot of plants that would have exactly the problem you're stating, that they do not keep their documentation either at an FD stage or even at a PLC documentation stage. They don't keep it right do they (GM/PLC)
2.7 [So other people like plant operations may be able to get a better exposure to....] I think so, I think that would be...(GM/PLC) [...the concept of an FD?] I think that would be a very strong spin off from the whole thing.

2.8 You've got the built-in revision control tool....[So you see potential for better maintenance. Do you see that as a guarantee?] No....No guarantees (EM/PLC) ...The real problem is the reality of it all, I mean, if they couldn't do that they'd bit of wire and hot wire the thing get the motor running anyway. The guts of the thing has got nothing to do with this thing actually, the guts of the thing is there's a motor not working out there and there's a production manager about to lose his job and all those sorts of thing and the technician or engineer will get it running and then he'll worry about this later...They'll say bull shit. They might tomorrow. I'm just saying, reality, the pressure that's put on those guys. It's very easy to be objective and clinical about it all but it won't happen like that. Not at 3:00 in the morning...(GM/PLC)
2.9 [If you had CASE tools and standards like this whether FDs might be shared more than they are now...Do you see any sharing of FDs?] Within an organisation perhaps, but....(E1/PLC)...Not outside an organisation...I think the tools could be shared around but I think the actual FDs are held fairly close to the chest quite honestly. (GM/PLC)
2.10 [I did want to back up one step to the automatic code generation. I had one thought, if I could initially generate some code, and then let's say I couldn't keep it maintained because of the lack of the on line changes, there is the potential that if I had the FD and had originally generated some part of the code and later on people had been making change, there's the potential I could have sort of a reverse engineer. It's kind of like your programming environment has the ability to compare two programs...well this thing could compare what was originally generated with what is there now but it could maybe give you a report, not just saying these rungs are changed, but it could go back and say this selection is changed, or this step is changed....] I'll tell you, well that's pretty useful....(GM/PLC) Yes (EM/PLC) People do lose themselves. Don't they?... It's be pretty useful because I mean that's what happens occasionally, because people make incremental changes over a period time, then faults start coming into the system, and nobody can backtrack what has actually happened to change a system that was running well 2 years ago to system that is continually faulting today...(GM/PLC) Does seem fairly difficult though. (EM/PLC)

2.11 [That's in the development phase. Do you see a benefit in testing and maintenance as far as code, data table, or message generation?] I think during the testing phase, perhaps if you got into a situation where you had a major change, which does happen within a selection, then being able to rebuild that quickly certainly has some advantages. Minor changes, well maybe even minor changes if it's easy to use and do you might say why go down and insert it into the PLC when I can regenerate it. The guy's going to go away and change his FD and I can generate that selection again and just carry out what's there... if what you are generating within the selection logic is getting to be within 95% of all the code in that selection, then it's going to be the potential there to just do it again is going to be quite high. It's going to be quicker. And you know that what you've written in the FD is what you got in the code. I suppose there's operates and that you'd have to consider.(SM)
2.12 [How about maintenance, do you see that kind of thing happening in maintenance?] Yes, that's a tricky one. [Compared to current methods...there're in there on line changing rungs and you can't shut the machine off and reload a selection, right?] That's exactly right. I think that's a difficult one to answer. Well obviously your limitation there is you can't on line changes you can't regenerate and import that in. Well, you can't do that at the moment anyway, so you are limited presently to on

line changes anyway so. Actually, one thing that you can do within A-B is you can do a rung by rung comparison onto files. So perhaps being able to generate the code into a file you are going to end up with this ladder logic and you have this ladder logic sitting in the PLC and you can do that on-line you can actually go in and generate...I forget what the report is called, but it basically tells you any difference between the two files, gives you the rung, yes, that would be perhaps with minor change/maintenance is going to be easy to find. [How would they use that?] Perhaps something got changed on line, perhaps for example like, I'm thinking probably more the reverse going back, in the middle of the night they didn't update the FD and made a change, you know that you can go to the FD and generate the code and then do a comparison between the two, that's going to show you exactly were the differences are. I hadn't actually thought that before, that could be quite a powerful tool for maintenance guys. [Yes, was thinking about that too, the ideal model that any change made to the FD and your are going to regenerate code. Because we can't take the down PLC down and reload it. In another domain it would work.] Right. Yes. [... but then your idea lets say they are making changes... they've been making changes and somebody comes in and wants to know what's been done... we can generate the original code from the FD and we have an anchor where that code came from, like we know it came from selection 101 or we know it came from step 2, and then if we did that comparison we might be able to not only say these rungs were changed but we might be able to say well it was in this selection that the change was or maybe it was in this step of that selection.] Yes, exactly. Yes, that's right, I mean... [The A-B software couldn't tell us that, but I might have a reverser.] Yes, that's right. Exactly, only it's taking it one step further. If what you get out of the PLC is what you had generated, if you can export in text to what you are generating, I mean that's there isn't it. But even if you can't do that, if you can generate the ladder out here and you've got a tool, a utility within the A-B software that's going to give you some output of the differences, perhaps there's no reason you can't look at that file and say, well this is more user friendly, it was step 5 of selection V, whatever. I think it's potentially automatic. (SM)

2.13 [... we were interested in the coordination or control aspects, if we had these tools would there be possibly more sharing of FDs, maybe across process engineers or across sites...we've touched on the operator maybe getting more involved... Would you see it even at a bigger, higher level?] Across plants, or across process engineers....I'm not quite sure, I could see the potential there for Anchor, say for example, or even Anchor Waitoa, to have a database (data base) or a standard set of types of selections or FDs, and be able to say get Joe Bloggs down and...Start from here. This is how a silo selection is done, or product transfer is done, or alarming is done and here's an example for you to use...Between process engineers, I don't know maybe it'll be driven by, maybe what I've just explained will drive that a little. It's hard to say, it depends on the personalities and their approach...Yes, maybe there'll be a data base, there'll a package and you'll get Robinson Automation in and give this package, go to it. All the sites will be using the same thing, maybe perhaps with some management there could be a data base of things in there. (SM)

2.14 [Controlling FDs, do you see improvements?] I think so, the history change control that you've got embedded in all the levels, as well as just the outer level, like it does go down quite a few levels, they do it at the moment at the beginning of the document, but you know, like when you, it's always sort of abstracted out from the actual document which is good for reading it, but if you're working on a change, or

why something has been changed, to see the path of the change, having it embedded down lower I think is quite good. (SM)

2.15 [Any other measures come to mind other than quality and productivity?] Mainly quality assurance standard levels. [Conformance again?] Yes, mainly conformance, you can say it is 95% correct or...Oh that's right, the other thing in quality, I think this history is going to be very very important in auditing and trace back. Especially when the FDs running, I guess you'd call it running, in the client and someone makes a change and the product goes out of spec because of the change to the FD and associated code, you know, throws it right back...to this. I think that history thing with an audit trail, traceability and auditability. (SM2)
2.16 [Well, one of the objectives was long term maintenance would be better if we had this idealised environment...] That's because you've got...if you're going to allow people to change the ladder either it's regimented they change the FD to reflect the ladder or they change the FD and it's done automatically, by reverse engineering principle. [lets talk about the first one, the regimented. Would the CASE tools help the situation as compared to what's out there now?] They should, but I don't think they will for the reason I'll outline. If something needs to be changed, I would be sure that the programmer would just go straight into the code and change it, whereas really what they should do is go into the FD, change it, generate that section of the code, to make sure that what it generates looks like what the change should be and then manually put the change in. But I don't think they'll work that way...So what's going to end up is there're going to key what they think the change should be there're going to put it and this condition in your FD and what that generates and what those rungs look like may be entirely different now. It needs to go on that forward cycle. [Then do you want to do that?] Oh, no, maybe you don't have to generate real code. Why can't you just generate the ladder diagram? Not the actual A-B PLC code. [And how would that be used?] Well, he looks at the ladder and he looks at what's there now, and the FDs put this other few rungs in, he'll say well shit that's a good idea, I'll do that. [Would you do that for the initial creation of the code?] Well you could do...Yes, I think that's probably a pretty good intermediary step, just generate the ASCII ladder ...just generate the ladder picture. I mean, they print that out, that's the working documents anyway. Just get someone to type it in, a typist.... Now you don't put A-B neumonics on, you put this intermediate layer I was talking about on....Ladder's ladder....[So they use this as a guide and....but if I generated the A-B ASCII export, they wouldn't have to even type it in, right?] That's right, Yes... I think that would be a really good step, actually, because in terms of straight in terms of acceptance... because if you can generate that accurately that's one thing that the guys saying, this is right. You know, sort of building up confidence... And you'll be able to use that for feedback for optimisation in your compiler. See what I mean....guy says instead of doing it like this, you should really just optimise that to this...[Like they would change that step and right on the screen we could show the ladder?] Yes, that's what I was getting at before, you don't want to generate the whole code, you just want to generate the change that's required. [So do you see that as a possible benefit?] Yes, well, I think it's essential to making the product run. I think it's actually a key component, because we haven't got time to download 2 megs of PLC code every time I want to make one change. Takes 40-50 minutes. [Do you think the operator would go through that little step of changing the FD and looking at the ladder and then put it in?] I have mixed feelings on that ... [Or, not an operator, I'm

using the wrong, whoever's maintaining the ladder. Like the software guy?] Yes. [That would be a feasible?] Yes, I would see that being...Yes, it's just got to be regimented doesn't it. [So you do see, even if I couldn't go directly into the PLC, you do see a potential in the maintenance phase by just even having this capability...] Yes. [You could say show me how this file is change, or regenerate this file of code, or regenerate this step of code?] Yes, exactly. [So we could have better long term maintenance of the FD with that sort of partial...] I'd hope so. (SM2)

2.17 [You mentioned reverse engineering...If I could generate the initial...up here in the creation phase if I generated the initial code for an FD and then later on lets say after 2 years there have been changes made to the ladder...which were independent from the FD, there is potential for a reverse engineer ...] Yes, and highlight the changes. Definitely. [But... right now the A-B software can compare two files and say this rung and this rung is different] That doesn't mean squat. [If we had this, the original FD, we could say this selection is changed or this step is changed.] Exactly, that is what you need to know. What I would see is in the FD or in the code generator would have sort of markers where your selections start and end, see what I mean? ...and when you reverse it back it would look for those key markers and the compare.... Part of the problem is that when they re-automate is that no one knows how the plant actually works. But they've got to re automate it, no one knows how it works...I'm not too sure whether you actually need the key markers in that case or you'd actually go through the code and actually spend some time putting the key markers in. (SM2)

2.18 [Comment on controlling FDs. Do you see the CASE tools as helping that situation?] Yes, well anything that provides better control's going to help, isn't it. [How does it provide better control?] Oh, easier to manage isn't it, I reckon, file...the file's easier to manage, backup, know who's got them, know who's got access to them. [How is it different from the word processor?] Just it's easier to do, basically. [So these CASE, are you thinking the CASE tools would just provide this mechanism that would make it...] Easier to do. [And is it just the user interface that makes it easier?] Oh, I think it's actually the fact that it is the CASE tool and you back it up....I mean, oh, it's [the word processor document] just another word processor document, its got WP5 on the end of it, who knows what it is, it could be a letter to my Grandmother, I don't know. FD file, oh, that's FD documents, that's different. [So it basically ... structures it...?] Yes, I think so. (SM2)

2.19 [How about maintenance. When you are in the maintenance phase of the factory. Do you see any benefit of standards there?] Well with this it wouldn't be so bad with straight out maintenance. Maybe simple timer values you would be able to read the values and feed them back in, sort of reverse logic. It could be problems when someone's asked to make a change there and then and they make that change and the only real way would be straight out into the ladder and then it's a matter of having the system there that either they do a print out of the changes they have made and mark them up and take them back and feed it back into the top end. [Yes, I wanted to follow up on that because what some people in computer science are advocating is this idea of is you develop the spec and automatically generate the code and then any changes you go back and change the spec and regenerate the code.] ... not when you go out to a factory that runs 24 hours a day maybe for 30 days without a stop, and yes, it is not possible to regenerate it. [You mentioned this idea about reverse logic. I thought about the idea that I if have the FD that I originally generated code and then if the guy went in there with the ladder programmer and changed the logic in theory

maybe I could take an ASCII dump of now what is in there I could compare it to what I know was originally there, and since I know the step structure, I might be able to at least say "Step 3 has changed" and this is what is different.] Yes, or even if it's on a straight out selection basis. But Yes, I mean the facility would be there to see it changed, but maybe not the change. [Do you see a benefit in that?] Yes, because that's not fairly easily implemented I don't think. Well it's a fair amount of work to be able to reverse it all the way back through the system, isn't it. ...Well you could compare to something you know... There are not many things that really change. I mean, once a plant's commissioned, and during the commissioning phase you are in a structured sort of, well maybe no you are, but you tend to have to be in a structured phase of what you're doing and then go back and rewrite the FD to come back down. [So, you do see some merit in the idea of being able to reverse?] Yes. (SM2)
2.20 [Well the fact that it is a domain specific CASE tool makes it more feasible.] Exactly. [Because that's all it does...] That's right. (SM2)

2.21 [Suppose that online changes could not be made to a production PLC...So the question comes done where, long term maintenance of the FD, is it going to be better, worse or about the same if I had this environment?] Maintenance of the FD if it was a full code generator without the ability to do online changes by the FD? [Right...or even if it didn't generate any code. Let's say I did generate as much code as I could, maybe I could generate 70% of the code and the rest would be hand-coded and merged. Do you think this environment would contribute to maintenance of the FD?] No, if the discipline was enforced could do, but I don't think it will because as with all people who are doing these front line changes in the thick of it it's just another thing for them to do in all the sequence of steps they've got to do unless that discipline is religiously enforced on them and it's just...I've got to get out the computer load of the data base rather than just scribbling a few notes on a piece of paper, it's more effort to use this, and it's another tool to learn how to use. It's all a question of management discipline I suppose. [Do you think if they had this tool as opposed to a word processor there might be a better chance of doing it?] Yes, a better chance, though not guaranteed... (EM)
2.22 [Thinking about Anchor's current methods, what would be the effect, or impact of using standards?] I think the main benefit would be the maintainablity. The greatest benefit I would see is that if there is a method in there to make changes as the FD is changed so it gets stored in a central data base. I can see benefits in that, because at the moment it's just written on paper if people make changes, to change the FD, if you've got the source code anyway of the FDs it's a big effort. Very soon it gets out of sync with the PLC. That's a practical application, where the major benefits would be I think. [Thinking about that, the current method is to write the FD using a word processor. Could they give those word processor files to the plant people, and could the plant people maintain that way?] Could be done that way. It isn't I think, well just a discipline we don't have in our organisation to use the existing systems. This would just be a way of maybe formalising it a bit more to, another tool, to make it more formal. (EM)
2.23 [If the CASE tool did generate code from an original FD and then let's say we go back into the plant in 2 years and they want to, lets say there's a need to make a change, I could potentially take a dump of the code that's in there, in the PLC now, 2 years in the future, whatever, and I could have my originally generated code automatically, and since I know the FD that code came from there's a potential that I

could have a program that compares the two sets of code and it could find differences, like it could say selection 101 has been changed or maybe it could even say step 1 in selection 101 has been changed, and maybe it could say well there's a new item that occurs that didn't occur or an item no longer is referenced, kind of a maintenance assistant. Do you see any potential in that idea?] Those comparison tools must exist at the PLC level, do they, A-B provide those comparison tools? [Yes, they do provide them but they can't map it up to the FD ... they can say program A and B are different...] Right yes...in these places, yes... To reference that up to the FD to say that it is selection whatever and...yes there'd be some value in that. [So that's a potential help in maintaining the FD. A reverse engineering concept?] Yes in maintaining the FD and to avoid any mess-ups if you are thinking you've got the latest version and load it in and you realise that you didn't have the latest version when you started making your modifications... You said give me the FD and they gave you a disk with the FD on it but it was perhaps 6 months and there'd been some changes. By loading that PLC stuff in would detect a whole lot of differences, save a few mess-ups, that sort of things do occur with many people trying to work on the same PLC with, make a small change there it gets very difficult sometimes to know who's got the master copy... (EM)

2.24 [I have a couple questions about the potential of maybe sharing FDs with other people in the organisation. again there's some overlap here, and maybe controlling FDs. Do you see any potential for sharing FDs within the organisation if there were a standard and there were in a set of tools like this?] Only no two factories are the same in their equipment...if you were building a new process line that was similar to another one, they would give you their FDs as a starting point for, again you could do that with the existing mechanism anyway. [Someone mentioned the idea that you could maybe have like FDs for just some part of a plant like just separators or just pasteurisers...] Yes. [The idea of having these little FD modules maybe that could be reused...] Yes, but again that reuse would be within the equipment supplier, they're always providing separators to all the factories, so they... (EM)

2.25 [In control, there's always the mechanism if it's easy to make a change then it gets uncontrolled very quickly, but if it's difficult to make a change then at least the document's controlled even though it's... [So ease of use could have a negative effect on control?] Yes, it could. In one of our factories one of the best plants which is <name> is because nobody every changes it because they haven't got the ability to change it, just keeps running as the manufacturer supplied it. ... But if they had the ability to get in and change it they'd have all sorts of problems. There's a trade-off there I suppose. (EM)

2.26 [Can you think of any other measures of effort or effectiveness that might be improved?] Probably just the history would be quite an advantage. [The history capability?] Hmm, trace back when you made a change and maybe why you made a change. (SE)

2.27 [We talked a little about controlling FDs, one of the goals was to control changes to FDs and control changes to FDs. Do you think the new methods would contribute to controlling changes to FDs over the old system?] Yes, I mean you could have an archiving system or whatever of whatever, history of when you made changes and why, and maybe sort of on top of that you may want to have one way of referencing all those changes...go in and have a look at all the histories of what was changed and for what reasons and you could generate a report of what was changed

and what date... like be able to search for changes after this date or between a certain range. [Yes, that's a good idea, so under controlling there's ... you are not controlling what they are doing, but you are able to go back and see what they did?] Yes. [And maybe search on that and generate reports, so it's more like reporting on what's happened.] Yes, it just adds that extra ability to monitor what's going on. (SE?)
2.28 [About long term maintenance. The way it stands you're saying it's probably not going to help. But if we had this reverse capability. At least we could say, we could find where changes were made. And that could be a benefit?] Yes. (SE)

2.29 [That's the development phase. Testing and maintenance, what do you think about that model of you change the FD and regenerate the code when you find a change. Is that feasible in this environment?] It's feasible to a certain extent. You wouldn't necessarily be able to down load that code. Because it might be running. But there's no reason, as long as it is good code, cut out the pieces that you need or copy the pieces that you need. What I'm trying to saying is that your plant might be running, you might not be able to shut it down, so when a guy, definitely you need to update the FD and generate the code but to actually apply it you might have to do some on-line programming... On-line you have to manually go to it.... Manually do it, there's no way to get that in there. You either have to shut it down and reload it. If it is a major change like you might be doing a project during the winter, which in that case... That's probably the biggest work you'd sit down and use this thing to create your FD, it could be a lot of work, then you generate the code and down load it. That's what I'm doing at the moment. Usually in the winter you've got projects on. (SE)
2.30 [It may just be feasible in the development phase or the major update phase, but not in this online testing, commissioning?] At least it gives you the opportunity to update your master, which means you can create a hard copy of your FD. He doesn't have to go back home and find the pages that have been written on with the green pen, find out what changes were made, you can be sitting there with your Windows up and say we'll add this check and your FD is up to date, at least functionally, that's up to date. Maybe just have a cut off once you've generated it. (SE)
2.31 [The question of long term maintenance of the FD. The theory of the idealised environment was that we'd have better long term maintenance of the FD because you'd be putting changes into the FD and generating code. But it looks like, at least on the surface....what do you think, do you think long term maintenance would be better, worse, or about the same?] Certainly it would be better because your FD would be up to date, because once Brian disappears off the job and systems guys on site and start making changes those FDs are going to be out of date. [They don't even use a word processor and go in there?] No, generally not. Generally they might pencil it into the book. But then you really don't have a proper document. [Do you think they would if they had these tools?] Oh, definitely. [Why would they use these and not the word processor?] Because it's for starts, easy to use, controlling it has always been a thing too, who's got the masters, I think it's something that's easier to control and it's easy to see what, the process is well documented, the process that its for, you don't have to be familiar with the format the guys written the FD in. You don't have to be familiar with the format the guys written the FD in. It's the biggest hassle, formatting the thing properly. [So just having the easier to use tool and the standard would encourage people to put the change in?] I think so. (SE)

2.32 [I was wondering if having these tools and standards would encourage or not sharing of FDs, reusing FDs other people have written, and I think you've already talked a little bit about controlling. Do you see any impact in this area?] May do so. FDs are fairly plant specific but they give people ideas I think mainly. If the plants are very similar then they can be adapted fairly easily... I'm not sure how easy it would be to change really... But I wouldn't imagine that it would be as bad because word processor documents pretty cumbersome, I think. (SE)
2.33 [We talked about controlling...] I think it'll be easier. Once the plant is commissioned you can hand this application over and then the site has control of their FDs and if something is easy to use, select, I want to modify the cheese factory FD, and selection such and such and poof there it is, much easier to work with than having to decide which document you're going to modify especially at the moment someone might use one document with 100 selections, someone might have a document with 50 sub documents in it . [Separate files?] Yes, separate files and different conventions in naming them. Very messy. With this thing it's basically the FD. (SE)

2.34 [How about maintaining and testing. Do you think it is practical to change the FD and regenerate the code as opposed to current methods?] We're doing the job twice at the moment...Cutting out the step of two people doing the same job is a plus but I'm not sure how feasible, I don't know whether I'm the right one, there are some things that are done on-line on during the maintenance and testing, and that comes back, you can't afford to shut down the process to make those changes...Once you've got the plant running, it can't, at least the process, you're method of getting that information into the processor has got to be transparent it can work with the processor on-line, could be a difficulty. (PE)
2.35 [Do you see any potential in what you've seen in the CASE tool to improve the long term maintenance of FDs?] Well, there is if it is a seamless loop, to change the program you change the FDs, you only have to look in one place, you just change the FDs, that would be the ultimate, then the FDs must remain updated. Unless there is something that enforces you to change this written word here, if it just remains a little book to describe what's going on then I guess nothing's going to improve it unless we can close this loop here and say this is where the changes are going to go. [Another aspect is ... the tools may help control changes to FDs or impose more control on what can be done with an FD whether that was an advantage? It's both isn't it as we saw before. Largely an advantage. (PE)

2.36 [A couple questions relating to sharing and controlling information. If there was and FD produced using these tools, can you see a potential for sharing FDs with other people or within Anchor, maybe reusing FDs written by other people. Do you see any potential in that regard?] Well, we do that now extensively. Nothing's secret as far as I'm concerned anyway if someone wants to copy my FDs they get it as long as Anchor's happy of course. Mostly it's Anchor that asks for them to hand on to somebody else. [They hand on the hard copy?] Yes, oh, and disks. We prefer to copy disks, there's no problem with that. And controlling FDs, that's a little more tricky. If someone does some changes on site and doesn't give you an updated disk of what he's done, he doesn't feel responsible if he's only done a tiny little part of it but you, although you do feel responsible, don't know about it so there's not a hell of a lot you can do and that's where some of this out of date stuff starts to catch up on you when people come and do little bits and pieces and don't clearly identify. We've come

up stuck on that a few times. Again, that should be taken care of with the ISO 9000 documentation on sites. But ISO 9000 unfortunately is a lot about documentation and not a lot about implementation...There's a lot of lip service taken place, which is unfortunate. So, yes we do have trouble keeping FDs up to date. If there's one standard method, if everybody had a copy of the software, here's new disk of the changes I've made. (PE2)

2.37 [What about when I get to testing and maintenance. What I'm about is that model of when you find a change you make it to the specification and regenerate code...about this potential of generating code...?] You make sense in two ways, apparently, I'm not an expert in it by any stretch. Make a change online, but the change you make online is volatile, as soon as you turn it off or go out or something the change you make evaporates and your back to what you had before, or you can load it some other way, which I don't fully understand, and then recompile the program, once it's recompiled then your changes are permanent. The recompiling takes 10, 15 minutes. So if you're in a commissioning situation, you haven't got 10 or 15 minutes. It has to be done yesterday. So the online changes you make directly to the ladder are by far the quickest way, of course. [Would you say common, that's the most common also?] Oh yes, directly online, on the fly. If you had to make changes through the FD system and load it down as it were, recompile the code ore whatever, that took time, then people would start to grit their teeth about that. [Relating to that, do you think these programs would contribute at all to the long term maintenance of FDs? I've been told that when you got into a plant after 2 to 4 years the FD is not necessarily very useful, because these changes have been made to the ladder over time.] That's because there's been no fixed, and forced requirement for people who make software changes to actually fill out the FD at the same time. That's changing with the ISO 9000 registration that's going on around the place and there're getting a hell of a lot better. Much better. But I still have to admit there're not perfect. Even the sites now that are ISO 9000 registered still slip up and don't put everything in the FD. [Do you think these tools would at all help that process?] Oh, must be. [In what ways?] If we can have a laptop sitting there beside the software guy with a FD like this sitting there, then he's probably more likely to go and go chuck it in. [But he could do that now in his word processor.] We don't provide that now, we just give him this, this one here...Could, but we don't. [Would you say this would be any benefit over that?] Oh, must be. Depends on how much they've used this. If they were familiar with that then it's going to be familiar there, whether than trying to learn how to start word perfect up and how to get the bloody correct file out of word perfect and what the tabs are set for and all the rest of the silly work that goes on in word perfect. So, probably that would be easier, because less errors. (PE2)

2.38 [Given the comments I just made, do you think these tools would contribute to the long term maintenance of the FDs?] Given how they maintain them at the moment, they've got to be better. The FDs are not maintained... I've gone out there and I've seen some like hand written, you know... It depends a bit on who the, I suppose which site it is and who the sparkys are, who the operators are, from what I've seen I don't think they are maintained at all. .[..so your impression is with tools like this, long term maintenance should be better?] Yes, if you can achieve your long term goals, they've gotta be maintained, they've gotta be better, because the FD is gotta be heart of the control of the plant. [Even if I didn't have the long term goal of completely generating code, would you see having this software available as a

contribution to that?] Yes, if were used. If the systems were there to ensure that the audits were there, the people were there to actually maintain it. (OE)
2.39 [Any other comments about long term maintenance?] I think I've probably said most of it before, really, if the FD is going to be the heart of the operation of the plant, if that full loop is there then the maintenance of the FD is going to happen. But if it's not, it depends on the systems which are in place to ensure quality... [Policies?] Policy. (OE)
2.40 [And the controlling FDs, keeping them maintained and controlling changes, do you think, we've probably touched on this before, but would the tools themselves contribute to that as opposed to what they do now?] Yes, there's got to be an improvement from we do already. Again, I think back to Te Rapa, the FDs which we have are all hard copy, for example, we didn't even have the soft copies passed over. So it's pretty hard maintain an original .. soft copy. It also relates to, you know, the FD's actually held within the control system... perhaps if the FD's held on somebody's, on the sparky's PC then, ok it might be controlled, but if it's held on a, an area which is pretty close to the process, I think you would have a much better of controlling it. (OE)

B.3 Organisational

3.1 I think that the way that this [CASE tool] is going through with creating some formats and standards and then being able to...I can already see that that sort of systematic approach, and standardised-based and rule-base approach would be of great value to us, wouldn't it? (GM/PLC). Yes (EM/PLC)
3.2 It is a structured form that really appeals isn't it because you're going to make the other 2 or 3 engineers use that structure. They maybe creative in part of what they apply, but ... not only that but it's obviously doing some of the work for you which is excellent. (GM/PLC)
3.3 [If we just had standard FDs, compared to current methods...would see an increase in productivity?] Yes. (E2/PLC) [For creating FDs and code?] And code (E2/PLC)...If all FDs were the same...we'd go through the learning process once...(E2/G/PLC)
3.4 [How about quality, would you see improvements in quality?] Should do...trouble is people would start to get bored, I would, just typing in code. That's not a problem, just a junior engineer.... (E2/PLC)
3.5 [Note: comment on need to be unique] and the thought and all the rest that's been proven on 100 other projects applies to him. He wants to be unique. He wants to go his own way. And wastes 6 weeks of time...so there's some interesting real world things in life, whether that's helpful or not. (GM/PLC)
3.6 [Do you think these tools are easy to use or hard to use?] I think they look great...(GM/PLC). [Do you think they'd be easy or hard to learn?] It looks like it was pretty easy to learn.(E1/PLC)
3.7 [Do you think these tools fit this particular domain?] From what we've seen does, but I mean you are basing it on a specific type of FD, that's what we're saying... Well it would fit the people at CSE that used to work at Anchor down to a T. And a few other people that used to work at Anchor that are consultants in the Hamilton area would be...(E1/PLC) ... if you go down to a site like Tasman, they'll tell you how they want to see their engines, their motors controlled, and checked and all that sort of thing, and it could be different from any other site. You go down to Kinley, and

they've got standards going back 50 years, and they'll say this is the way we want you display a motor and the way we do it, and that might be close to Tasman, because a lot of the people at Tasman came from Kinley, that's the only reason it would be somewhat similar. So its been a quasi-standard developed by some engineers back 30 years ago. And you go to another site in a similar industry and they just abhor the way that Kinley does it, don't they....so it gets right down to some professional, intellectual things within engineers going back many years, I believe, don't you? [The big issue we're asking is, is a domain specific tool like this of more value than a generic one. The generic one takes away the specificity and adds some general kind of features. What we're really trying to say does this have benefit for Anchor and does a similar version for Tasman, and another version whatever, are all those versions individually better than one generic one that kind of falls short in any areas. What do you think?] Well, nobody likes get fitted up with a straight jacket...If you were in the business of providing these customised systems, presumably you've got the basis of all of them there...(EM/PLC)
3.8 [So you're saying that even within a domain specific tool there still needs to be these customisable rules to make them adapt] Yes...(EM/PLC) ...well the idea of steps might not be everyone's preference. And for each step, precondition, operate, check, I mean they might not want to do it specifically that way...it sort of, like I was saying before, dictates a style of programming, so that if you don't think about the whole process and you just start reading here, and oh yes, one of these, and one of these, and one of those....

3.9 [Compared to the current methods would you say that these tools are easy to use, harder to use, or about the same as current methods?] I think it is on a similar ground to that. From what I've seen it's all pull-down menus and pick and place with the mouse. No, I think it is on a comparable level. [How about with the fact that it's using the structure and terminology, it's basically using and FD structure and terminology that came from the industry rather than buying and off the shelf CASE tool or something that would ask you do use DFDs or something?] Right. I think form the point of view of using it as a tool to write FDs for the dairy industry the way everything is described is probably better than what you'd probably get off the shelf, that's standards, as you say a standard CASE tool. (SM)
3.10 [How about learning, would you say it easier to learn, or more difficult, or about the same as current methods?] Similar ... it's quite nice and modular and the structure is similar in each one. (SM)
3.11 [Do you think that the tools fit the need?] I think, it does, if the ultimate thing is to generate code automatically at the end of the day then the tool fits certainly that need and you could possibly just cut the tool off at the FD point and use it as a useable tool just right there. Absolutely, it does fit the need, I think. Certainly seems to have fairly comprehensive set of tools there. (SM)
3.12 [How would standards affect your effort, your productivity to create code, or we can look at some other life cycle phases, like test or maintain, or how would a standard FD affect the quality of your code as compared to current methods?] I think without a doubt, with a standard FD that the productivity would take a lot less effort to produce code without a doubt. I suppose a lot of problems we have at the coding end is getting different representations of the same thing and then having to decipher or understand what there're asking and then actually implementing, you're seeing two

or three different ways of doing the or writing the same thing. I certainly see that, a standard FD of helping us there. (SM)
3.13 [Less time to create the code?] Certainly, with out a doubt. And getting to the quality type thing, the quality of our job would improve as well because we are working from a consistent presentation of information and therefore our, what we're producing would improve as well. We do spend a lot of time at the moment communicating with the process guy, which is good, you know, it is essential to getting the job and done well, but a lot of that stuff is just dealing with the finer points, what are you really asking for here, sometimes you might have a whole lot of conditions and checks and stuff but it's not really clear what there're actually to achieve or the, it only takes one little word to be wrong and what we're trying to implement is blown out the window sort of thing. [So you think a standard would be less ambiguous?] Certainly. Absolutely. I'm biased, I'm quite a believer in standards. There certainly can be situations in which standards can work against you as well, but having come into Anchor and only been within the system for a couple, three years, it's something for them that is certainly lacking I feel. There's a lot of benefits. For operators as well, they see the same document.
3.14 [Operator's productivity?] Certainly, the ownership thing. If they see the same thing again and again with improvements along the way they accept it straight away. Whereas the might pick up the FD that I have written, and they go, just because I've used a different font or formatted it differently, it says the same thing but there's a lot of resistance. It's quite surprising, just watch the reaction of them...I think because of the familiarity and the way it's structured I think their learning, would, their capability of sitting down and understanding what is happening in a selection would happen quicker. [Easier to learn, faster...?] Sure, because they understand the structure and what, I'm sure there must be quite a step there when they get a new guy in with a different method, different terminology, even it is only like one term for a state of something is different, it could confuse them quite easily, or does confuse them quite easily. So yes, I think their training and stuff like that would be easier and would happen quicker and have, the result would be achievable (SM)
3.15 [Would day to day operations improve, do you think there would be more efficiency there or less errors?] I think the ability to, if they, different plants work differently, like some operators will have the FD there all the time, and if they are having a problem they'll have a look from that point of view they probably look at it when they are having a problem. So if they are familiar with it and there's a standard structure my impression is would be that they would be able to sort out a problem a lot quicker for sure. It's just familiarity is the biggest part. (SM)
3.16 [How about in the testing phase, do you see a benefit to standard FD in the testing phase?] I think once again the familiarity thing would come through, it would enable, one of the problems we have when we are testing and simulating is that you might not be able to have the same guy that coded the code, and it might actually be better that it's not the same guy involved with the process guy...If it's a standard, I think. It's familiarity again. I think certainly that would help us and also perhaps there's, I think, a possible benefit when you are building the simulation, or the testing, you do actually have to spend some time building up the simulation thing, if the FDs are standard then you are going to be doing, checking the same things, going back and checking things in a similar manner...[that's when you build that simulation software you have?] Basically you've got to set up at the moment, you've to set up ladder to emulate field I/O and you may also with analogues you build up tables of

values that you can manipulate, you know if you are dealing with similar or the same approach each time then you don't have to go and change your methodology.
3.17 [How about maintenance phase, after commissioning, do you see any benefits, improvements in productivity or quality at that point with a standard?] Once again, yes I do in that, once again, the familiarity thing...they know the structure, they know where to go to edit things...And certainly having a standard structure has gotta make that, and it forces them to obey or work within the bounds of that standard, which if the want to change or add a step they are going to do it in such a manner that the guy that's going to have to implement that's going to, you know, and also coming back maintenance once again if some code change does get done in the PLC then hopefully because of a standard structure then hopefully it is going to be easy to reverse engineer that back, hopefully we're eliminating too much of the go and code it and go back that way, hopefully most of it will come the other way I think the production people are actually starting to appreciate the value of writing up as a FD the changes before they try to do...I think a key thing there is the operators doing a lot of maintenance of the FDs, rather than getting the Murray Dench's in or employing somebody for 6 months to go in there and try and sort them out. (SM)
3.18 [As far as the overall organisation, then, you do see benefits across the board?] Yes, I do, all the way through the whole gambit fright starting from the I/O, writing FDs, for us coding, and the guys in the field actually maintaining it. For sure. (SM)
3.19 [OK. Do you see and drawbacks to using a standard for the FDs?] I suppose the only one possibly that could come in is when you are trying to deal with a process or a situation that perhaps doesn't fit the mould, but overall, no to me the benefits far outweigh that one. That would be the only case I think you could get some...but then if you sit down and think about it I think you can work through it anyway. I think that'd be the only one I'd see. (SM)
3.20 ...I think the tools, it's certainly a better set of tools, or a set of tools that are customised to the application, as you say, giving it a fit...You know, even just coming like having my background as being electrical engineer and coming from one industry to another, like from the pulp and paper industry to the dairy industry I mean you could stand back and say they are both continuous types of process, you are dealing with similar sort of, you know you've got the pulping, it's a fluid product, and all that sort of thing, but jess the differences, you get down to the automation, you are using the same hardware and that sort of thing, but when you get down to the FD and the code it's just poles apart. And I don't think, I certainly don't think you'd be able to apply the selection type, or perhaps you could, but I think it would be a lot more difficult, to the pulp and paper industry...I got quite a shock actually, it's quite different approach. Mainly perhaps because the dairy industry is maybe even more automated in certain aspects of the way the approached it, or the way Anchor has approached it is perhaps is what the difference is. Anchor has had a fairly unique methodology. The way they have proceeded in the last 5 or 6 years has in some ways driven what's happened in the dairy industry in NZ. (SM)
3.21 [As far as long term maintenance of FDs with these tools, would you see improvement, better, worse?] Better. [Better in terms of...] I think ease of use. [Easier to maintain?] Yes. [Better quality ?] Yes, empowering the user to, give them the tools that are going to make it easier to do that. (SM)

3.22 [Do you see just having a standard FD as a productivity improvement for the operators?] Well, yes, certainly, [Drawbacks?] One major: Stifles innovation.

That'd it...Oh, it just kills it...Well if you want an innovative idea doesn't match the standard, doesn't go in, that's it. [So is that enough to kill the standard right there?] Depends on how big the idea is. They cost money [also]. [Cost, where is the cost?] Oh, just conversion and people getting up to speed. The initial cost and sometime your standard are not necessarily the cheapest, the most cost effective. Oh, it's like we have a standard on Compac computers now, you can do the job with a cheaper computer but that's the standard. I mean, standards cost. [But then some of the potential should outweigh some of the cost? Like the training or...] Well that's the balance of standards, isn't it. But you give me any standard and I can find something that'll be cheaper. Undoubtedly. And you're always going to have that argument. (SM2)

3.23 And the other thing about standards is that you can't necessarily just impose a standard on people that don't accept it. We've been doing it this way for 10 years, there's nothing wrong with our way, we use this type of PLC, why should I use your new one, my whole plant has Modicons, why should I put an A-B in....you know...you get those sort of arguments coming up...(SM2)

3.24 [This gets back to an earlier comment I made to you about generic vs. domain specific tools...we've developed these programs just for the dairy industry or Anchor. And I was interested in people's perceptions, compare it to the current methods, whether they thought these tools were easy or harder to use, easier or harder to learn, etc.] I think the tool is specific to Anchor in the dairy industry, I'll actually go as far as that... Don't know. Everyone can drive a word processor, but then you can type rubish as well. There'd be a learning curve, but I think its just like any package really to learn it. [Would you say that maybe easier or harder to use than the current methods?] There's an initial learning curve and then....Then I think it'll be easier. (SM2)

3.25 [Getting back to standards themselves. Anchor could adopt a standard but if that standard is still just handed out in hard copy it may not be maintained.] Right. [So not thinking about having the tools to help, do you see a standard helping in the maintenance phase at all? Let's say they adopted a standard but the standard still came out in hard copy.] Yes, I'm sure it be...each equipment supplier that we use at the moment has their own mechanism of transferring that sort of information and so, both maintenance people and programmers it's difficult for them to work out what's required, so if everything came in the same format, then for sure, but it's a discipline that we've got to enforce on the equipment suppliers, the suppliers of the equipment, because its their responsibility generally to generate those documents. [I think you're saying, like Brian Robinson works for the equipment suppliers.] That's right. They produce those documents. We can specify the format how we like them. And we've gone to that stage now where building it into, documents are standards in conditions of contract we give to the suppliers how we want it so we have gone to that step now with FDs saying you will do it like this. [Yes, you did mention the idea of if the programmers were given a standard you would see their productivity improve as far as creating their code.] Yes. [And do you see an improvement in testing if we had a standard FD, in either productivity or quality?] Not so much I don't think in the testing phase as in the creation phase. (EM)

3.26 [Can you think of the effect on productivity or effort as far as other people like maybe the operators or anybody else within the overall scheme of things?] I suppose yes, in fact with the FD document is that sort of user manual that they get them to run

the factory by, so if it always has the same look and feel no matter where there're in the factory has got benefits. Its something that's historically haven't been very good of but the FD should be the operator's bible and when things don't work he should be referring to that trying to work out...so having a standard format for that would probably be, the operators would certainly appreciate that. (EM)
3.27 [Do you see any drawbacks to using standards?] As you say, the loss of creativity is the only drawback... [Standards could constrain creativity?] Yes, that's always a catch isn't it. (EM)
3.28 [Any other comments on standards in themselves that you can think of? Do you see better quality coming out of PLC code as a result of standards?] Oh yes, for sure. There's certainly quality comes out of code reusability I think if you can always use code that you've used before or has been generated by machine and then you know that it works then have a much better chance of a system that works in the end with a less painful debug procedure. (EM)
3.29 [We are interested in, compared to current methods, or even potential generic tools, what your perception of these tools are in terms of use and learning...do you see these things as easier to use or harder to use or about the same?] ...what we've got here I think is reasonably specific to the dairy industry I think, or dairy/brewery/sort of food, it wouldn't apply to a petro-chemical plant or anything like that, but I think it is generic enough to sort of say food processing type, fluids that you're pumping around....I mean certainly the interface that you've developed is very easy to use by all accounts, it's quite straight forward... [Would you say it fits the need as it's developing at this stage?] Oh, yes, certainly for the having a maintainable data base that you can enhance during the life of the plant. (EM)

3.30 [How do you think standards might impact your effort, for example, in creating in your case PLC code. Do you think having a standard FD would affect your productivity?] I'd say it would go up quite considerably. At the moment I think more time is actually spent deciphering what the FD writer wants. A certain amount of time deciding whether what he's actually got on paper is achievable because maybe he hasn't thought of some of the limitations that are there so you'd probably spend more time actually writing the code than actually thinking about what or deciphering what he's trying to do. [...the word that popped into my mind is ambiguity. Are some of these FDs ambiguous, you are not sure what the FD writer meant?] Yes, in some ways its ambiguous but...things like when he's telling you to do something and sometimes you're not always sure whether he actually means what he says so its not really ambiguous, but it's maybe just determining what he wants. [Just interpreting?] Yes. [So a standard would reduce the need to interpret or go back and ask him?] Yes. [So you think it would save time?] Yes. (SE)
3.31 [How about quality. Do you think it would impact the quality of your code?] It would have a consistent approach about it and maybe one thing about it is as long as the ability to sort of redesign the top end when you've decided well maybe you wanted to add this functionality into it as long as you could change the design to accommodate improvements and enhancements then it would improve the quality because someone came up with an idea -- and you think yes it's a good idea -- why don't we do it that way for some reason and that's maybe sort of...there's a structured approach in the way that things would happen. (SE) [So the standard would structure the way the changes may be handled and that would lead to better quality in the code?] Yes. [Do you think less errors?] Yes, less errors because if you are doing

one particular type of thing you are doing it the same way every time, you're not trying to do something a wee bit different. (SE)
3.32 [How about the testing phase of your software. Can you see benefits in productivity or quality there resulting from standards?] Probably the more experienced, the more experienced you are, just by reading an FD in a printed form you'd be able to see sort of errors or likely errors or likely problems, but as far as FDs, the debugging at that stage unless you actually see it actually written down then you would won't really see it when you're testing it to some degree because you might not appreciate what's faulting and you might not test the right thing anyway. [So you're saying that if your are going to catch it in the FD you would have caught it in the creation stage.] Yes, or the initial read over. [And then testing it is going to show up some other way. Not in the FD...] Yes. (SE)
3.33 [Now I'm going to ask you about other people's productivity as far as using standards. Or the overall organisation. Do you see benefits or drawbacks to people other then yourself if we had standard FDs?] The only drawback I can see is where maybe you've got an application that you can't fit easily into the standard and then you could be spending a lot of time just fitting it into the box... Most of the time most things are pretty straight forward but when you've got things get quite complex... [So one drawback is if you have some process that was a little bit different, that didn't quite fit the mould?] Yes, but probably 85% of the time it would increase it. [85% it would be better?] Yes. (SE)
3.34 [Do you see a benefit to like factory personnel, the operators on the factory floor?] Yes, I mean if it was totally in Anchor, that was their base-line, the operators, some operators may not be that good at reading FDs but once there's a structure they are familiar with using then they can immediately go to that step and it would mean something to them because they are familiar with it so yes. (SE)
3.35 [I want to ask you about the tools themselves. One thing that we have done with this set of tools is we've developed it just for the dairy industry, but more than that we've developed it just for Anchor...We've tried to make it fit. And if you compare that to the tools you use right now, your current methods. What do you think of the potential of this? Would you say it would be easy to use, or difficult?] Yes, at the moment it's a big job to try to even get the FD writers to change a format for that reason. That's sort of a bias of the PLC but when you can't see the reason. But if you load it from the top end then that's the way it is then it would sort of help everyone who used it and it would help quite a bit. Because the formats there you'd spend more time...I think the key to what you are trying to achieve is to get to operate the plant on automatic and get as much information to the operator, so if they were trying to think about all the conditions and bring all information to the operator and those were the two things they were thinking about rather than the format then they could use their time better and the end result would be better because they weren't thinking about the format so much but what they are actually trying to achieve. [So you are talking about the FD writer. There wouldn't be this discussion of how to write the FD so much as you move onto the important things.] Yes, what you are actually trying to do with the FD. (SE)
3.36 [How about using these tools? Do you think they are ease to use, hard to use, hard to learn, easy to learn...] They look pretty easy. Just in some ways being able to maybe edit and copy a complete FD all in one run rather than step by step or whatever. Just sort of enhancements on that...I mean if you can copy a selection, that's fairly powerful but if you can copy multiple selections. Just the drier in the

Waitoa factory there's 3 evaporators, and basically there's maybe 40 selections for each one and there're all similar in some respect. So after you've got one if you could tag and search and replace. (SE)
3.37 [Do you think the tools fit the need at Anchor?] Yes, I do, it might do me out of a job, might have to look at something else to do..(SE)

3.38 [So that's my first question, how do you think adopting a standard would affect your productivity?] Oh, well, greatly because having a standard FD means you can have a standard PLC structure, the database structure, and when you've got that, when yo2u switch between processes, different FDs, you know the format you're going to use... (SE)
3.39 [Thinking of the FD itself, if your FD followed a standard...] The standard removes ambiguity, not having to interpret what the guy actually means. [Would you say it would save time in coding, to have a standard?] Yes. [How about quality, do you think your code would be a higher quality? About the same?] I think it would be higher quality because if the standard was the same then it would get improved, instead of developing a system that works well, then having to change it and having to interpret what something means. [Because its ambiguous?] Because of ambiguity, Yes. (SE2)
3.40 [Can you think of other affects on your productivity?] It's just the standard, that's the main thing for the process people it's the same for them, they know what the standard is, add to an FD using the same standard format, it's easy for them to follow. (SE2)
3.41 [How about the testing phase, can you see benefits in the testing phase, to having a standard FD? As far as you're concerned as a software guy? Can you see productivity improvements in the testing phase or quality improvements in the testing phase?] I don't know that it would have that great of an impact. [You think that the benefits are mainly in creation phase?] Yes, and the interpretation. [But you've probably worked out the interpretation in the creation phase by the time you get to testing...] Yes. That goes hand-in-hand. Once you understand it, you're familiar with the standard then the testing is .... it follows right through ... you change a statement you learn, you interpret, follow right through... (SE2)
3.42 [Can you see any benefits as far as productivity in the maintenance phase to having a standard?] Yes, the same really. Knowing a standard, if you have to add a selection then you know how to do it because standard, you don't have to, what we do now we look at 3 people's FDs and find out what we like best and have to determine how to write it. (SE2)
3.43 [The creator is not the same guy as the maintainer. Is that true?] Yes, definitely. [And so, if there was a standard across the organisation, that guy that's inheriting that code should be able to understand that FD] Definitely ... You could be some process people in to help you write your FDs, there could be 3 people working on FDs and the result to the coder, can use it equally as well ... And that point about handing the code over once the system has been commissioned, then it is a different set of people that are maintaining it, that's an important point. (SE2)
3.44 [Thinking about the productivity of other people, do you see any benefits to other people in the organisation... as far as having a standard?] It's the same really. By having the standard, you don't have to reinvent the wheel, you don't have to, a new person on the process wants to write a function description, might come up with an idea to put a piece of plant in, and would be asked to write an FD for it, and then

the next thing they have to do is find out how to write an FD, and it just seems there're so different, the documents are always different, you always have to interpret, now what does this guy mean. So by having a standard, then it's something making reference to it, it helps them write it and it removes that ambiguity. (SE)2
3.45 [OK, the operators, they guys that are running the plant, they use the FD, is that true, to trouble shoot...?] Yes, to find out what their plant supposed to be... [Do they move around?] They do, within a factory they move, well they do, a guy is not always in one factory. [So they moving from this FD to another FD?] Definitely... As different as chuck and cheese. [So a standard would help the operator too?] Definitely. Some FDs are quite difficult to follow when you are used to one thing, move into a plant and have to learn another system, it's time. (SE2)
3.46 [Can you see any draw backs to standards?] Not really, as long as the standard does, as long as it's developed with input from people then I think it should be applicable, and that once its been developed it could be updated itself, that's pretty important, I suppose it's not necessarily very easy... It all goes into, like the ISO standards, we really have to have standards and this is another phase really and there's always a facility in standards to modify and update. I suppose it has to be considered. [Do you think a standard might limit somebodies creativity?] Yes, there is that, that is possible, there are some FD writers are fairly vague, they are basically giving you a description, whereas by nature you really need to be fairly explicit to be able to generate code from it which can be... you really have to explain it in an FD. [Would you say that's a positive?] Well, that's both. It's a positive because that all removes all ambiguity... Overall it'll be good, good for the company. (SE2)
3.47 [Would you say that in comparison to the current methods, would you say that these tools are easier or harder to use?] I'd say easier. For starters, we really haven't got any tools. (SE2)
3.48 [How about learning to use the tools. Do you see a difficult, or easy...] I think it is relatively straight forward. It's handy to have people like you here who have designed it to know exactly what you've intended at each step. But windows environment is fairly intuitive. Tools, that sort of thing. And using a word processor, that in itself is a cumbersome exercise, formatting headers and footers, and all.... That can be a bit of a pain. [Would you say that, thinking about those problems and learning these, you don't see a difficulty in learning the new tools?] Oh, no. [Maybe easier than the current?] I think it should be easier... And one thing with the word processor is that invariably you're after a report at the end and half the time is spent just making it look right. You've got to get the tabs, and you go through the document and do something, a format or something, you can spend hours just getting it to look right and getting it formatted correctly for a printed report is pretty important. That's not easy... Getting information from the hard disk onto the paper has got to be one of the more difficult computing tasks. (SE2)
3.49 [As far as the fit to the needs, would you say these tools generally fit the need?] Yes. (SE2)

3.50 [Potential and limitations of standards on productivity?] I'm enthusiastic as long as we are carefully looking at those standards... Even if we made the writing of the FD part a bit slower the overall task...the overall black box...the bigger box, productivity should be higher overall. (PE)
3.51 [Potential and limitations of standards on quality?] The standardisation may improve the quality because there will be less room for ambiguity because you are

forcing conformity even myself what I write if it no harder for me or Joe Bloggs to do it, then if we both confirm, then there is certainly an advantage. There are some tools in there that force you to think in a consistent fashion. As long as, you've got to get it all right. (PE)

3.52 [Thinking of other people in the organisation than yourself. Can you see positives or negatives of standards?] Yes, A major one because, ...I've had other operators who have used some other standard and say why don't we do it this other way. If there's only one standard I'd probably say yes right, Here's a standard and it's worth doing and the same for operators and other people. Something that's standard, I'd invest my time. That's got to be a major plus. And if other people have invested time then you'll get feedback from those people in assisting this part of the creation so I guess that assists this productivity that is a big one that I didn't think of when I was thinking about my self (where do I fit in) but if I can get feedback from people that are not normally used to thinking of FDs but know the process better than I do, then I can get their expertise, everybody else's expertise. This project here at the moment, the FD has been written by me and I'm not the best person on this process because I'm an expert in this particular tool and the way we do things and it provides consistency within the organisation and I've got a consistent way of doing it people who know this process better than me have to a totally different standard, the Dutch people , who have been used to a different automation system, they don't want to invest their time in this standard but if there was one standard they may, they may still give the job to me but there would be less problems in terms of I didn't mean this I really meant that. (PE)

3.53 [Can you see drawbacks to a standard?] Yes your ability to change. If the standard enforces something which is not desirable for what we want to achieve and forces you to work in a way that is not your aim. For this particular job...there was a standard that the programmers had developed from their previous experiences at another job which I guess it was related to the fact that it hadn't been defined on paper they endeavoured to write their code according to their standard to meld in with a FD which was actually different and the result in my belief is that the worst part of the program in this particular plant is where a standard was tried to be enforced...But this, what we're doing here, will overcome that because it will be defined and before this is written it will explicit we can talk about what those standards and say we'd like to change, we need a method to change the standard to improve. If you change from AB to a different processor how achievable is it, or are you not going to take advantage of opportunities because the cost of changing to a new standard is too high. (PE)

3.54 [If you think about your current methods, like your word processor or your other tools, would you say that new CASE programs are easy to use, more difficult, how would you characterise them compared to your current methods?] My tools at the moment are word processor. So you are comparing sophisticated word processor with tools which are less sophisticated but have other advantages. It is slightly hard to compare I suppose... I think that yes, they are probably better...Overall much the same...[Do you think the tools fit the need?] Yes I think they do. You've got a consistent means of approach and I feel that seeing what you've done here that I'd know what to do in half an hour or an hour. There's no problem in learning what you've done. In terms of understanding and learning what you've done there, yes I think it marvellous. (PE)

3.55 [Having the standard you were saying might allow other people in the organisation to learn and then standard and communicate and cooperate at a higher level?] Yes, more people would probably get involved in the FDs and understand, I suppose if you really think about that makes a big change there is a big advantage in terms of people will start looking at the process they won't wrap up automation as a black box, there is a fair amount of that going on, it's just too complicated to understand, I'll just push this button and that's what's meant to do, people may think if I push this button I know what's going to happen because I can read this FD. There are a lot of operators who may be in that situation. It's too difficult to understand I won't try to understand. I wouldn't like to be in the position of not understanding what happens when I push a button. (PE).
3.56 [If the SEs were getting consistent FDs the would you expect better quality code or less errors or misunderstandings in the code?] Yes, definitely. When I think back to where I've had problems, I try and be as consistent as possible. In general most, a lot of the problems, you look back at it in hindsight and say ah yes, that's how they've interpreted it, yes I haven't written it, I've been ambiguous in the way I've written something. If it can only be interpreted in some way. Because checking it becomes almost too difficult to do, total checking in the time that's available. (PE)

3.57 [My first set of questions is just the idea of using standards.] I standardise as much as I possibly can, including my messages. All the project I've been involved with I have generated the descriptors for the I/O....[...if you're doing a job for Anchor, have they already specified those descriptors?] No, it's my standard. Mine alone. [What I was going to ask you is, if Anchor imposed a standard on you would this affect your productivity in creating FDs, would you say your productivity would be impacted by that?] Yes. I don't like this way of doing it personally. Probably because I'm so familiar with the way I've done it and it's difficult to pick up a new method.... I write my FDs largely for the operator. The problem with that is you can easily leave the software guy behind...The difficulty with the way I do it is making it so that the software guy can clearly understand what you mean but still keep it simple enough for the operator. [Would you say this one [new] is more software guy oriented?] Absolutely. [Do you think once the learning curve has been overcome, you would be...you think your productivity would still be lower?] Oh, I would doubt it. I hope not. [...so you think once the initial learning curve, then you would probably be about the same productivity?] I would say yes. [Would you say that would fall across both creating testing and maintaining?] Yes, I would say so. (PE2)
3.58 [Do you see it impact on quality, the quality of your FDs if you used a different standard?] No. I go to extreme lengths with my FDs to make sure they are right when I put them on a piece of paper...If it takes me an extra day to sit down and do a selection, I go through it and through it and through it until I've got every bloody option covered and that's as perfect as I can make it here because this is the easy part. [So as far as a standard is concerned you're saying you would be still use your same level of thoroughness or whatever and a standard wouldn't get in your way?] No, so long as...no, not once you got familiar with the terms and the layout...when I go through these things, mind you I've just written the thing so I know them backwards anyway, it's just a matter of going through line by line and keeping the things that hit me over here in my mind when I'm going through this one...I would imagine that it would be no different. It's just that this system here is probably a little more impersonal. It's more structured, you just go click and pushing buttons, whereas this

one writing it down ..in my head, and perhaps you're closer to the ground doing it this way than just by punching keys on the keyboard, possibly. (PE2)
3.59 [So, just to summarise I think what you said is you think you would have more effort, but it would be initial effort, it would be like a learning curve to adjust to a new standard, and then once you're adjusted, probably your productivity would be about the same and your quality would be about the same?] Yes. (PE2)
3.60 [Thinking about other people's productivity, other than yourself, like maybe the software guy or the operators in the plant do you see a standard affecting their productivity?] If the software guys are being fed a standard layout, its going to be a heck of a lot better for them. By miles. Like there's untold different ways of writing a function description. And feeding into a software guy who has to handle all these layouts and different configurations, that's gotta be difficult for them. If there're going to be fed standard layout from all these different people, it has to be more effective. [More effective in terms it should take him less time to write his code?] Yes. [Do you think there would be less errors?] Yes, there would be less errors. [...better quality ?] Yes. (PE2)
3.61 [Can you think of any other measures of improvement?] You do it faster and more accurately, and that's got to be better for everybody. (PE2)
3.62 [Any affect on the plant floor people?] It's quite difficult for me to say. In the sites that I've designed and commissioned I rely heavily on operator involvement. Like I'm on site and we go through it, I live with the operator, talk with the operator. And teach them how to use these things. [During commissioning did you say?] Yes. When I've finished a project, the operators are pretty darn familiar with how to read my FDs. They use them all the time to learn the process. There're going through and saying what happens next, what happens next, what happens next. I've only been to one site commissioning where they've used this method and the operators never ever, to my knowledge looked at a FD. Never opened them up. Now I don't know if that because they haven't been encouraged to or if because they had done and couldn't understand it. [If there was a standard at Anchor, lets say it was this standard, do you think they could understand it?] Well, yes the must be able to understand it if you trained them in it. Yes, they must be able to. If...Other people understand it. (PE2)
3.63 [If they [operators] were trained in a standard then would there're job performance improve somehow. Would their productivity be better, if the operators all have the same standard and the software guys all have the same standard?] At the factory level I think it would probably come back to the commissioning guy. And at the time and if he is prepared to put into the operator at that level. See for example, you take that, if I said to an operator is flag C204S active or not, he wouldn't know what the hell I was talking about....He wouldn't have the foggiest idea...That's the difference in talking in operator language and in computer language or software language. [I was concerned about that...I put it in like this, CIP 203 Not Selected....Do you think that's a little more meaningful?] Much better. [It sounds like what you're saying is for the operator, if the operator is going through the commissioning with the commissioning person and there're going through the FD, he's going to learn it whether it's a standard or whether its not he's going to learn it in that process and so you think it sounds like the standard is not going to be that much an affect on the operator because there're going to learn it during the commissioning phase anyway. Is that what you're...] Yes, but of course the more simply its written the more simply there're going to pick it up. And the more there're going to understand...A site like Te Rapa for example doesn't have an FD. Don't have them.

The operator trouble shoots directly off the code...It's not ladder, it's para code. It's an English language type programming tools where if you say eng V6 and it means open V6 and deng means close it and all this sort of standard sort of English terms. But the operators haven't got a hope. They don't understand it one scrap...The idea was to have a few simple English terms which explain everything but as the process got more and more complex they had to add more and more sort of technical terms to make things work like move for example, moving a number from one register to another, all those sorts of things. Well the operators haven't got a hope. The see move R6.3 R5.8. They don't know what's going on. [What kind of a PLC is that?] It's ACOS. (PE2)

3.64 [My other question was the overall organisation. Would having a standard FDs have a positive or negative affect on the overall organisation in terms of productivity or quality or some other measure?] Must have. If everyone is talking the same language it must be better, it has to be. (PE2)

3.65 [Can you see drawbacks, to standards, negative, down side?] As long as, the only thing I don't like about it is if it anyway reduces the flair of individual people, like doing things in a smarter way, selections in a smarter way, anything like that. If it quenches lateral thinking it's bad. [Yes, if it gets in the way of innovation, or creativity?] Yes, and standards can be quite constrictive of course, by virtue of their being...And that can be unhealthy in the long term. But if there's still room for adventure and enjoyment, then...Because we are all in business to enjoy ourselves to some extent, if we are not enjoying ourselves what we're doing we shouldn't be doing it. There has to be enjoyment in writing FDs for me. Although it's boring as hell actually creating them when you walk away from a factory that you designed and built and commissioned and handed over that's a brilliant success it's a mighty good feeling...That's what keeps me going. If I was just a number in a wheel that's just turning a cog and I had no real feeling for it all of course that would reduce my enjoyment to no end. (PE2)

3.66 [Would you consider these tools easier or harder to use then your current methods?] I've got to say they'd be easier....They'd be quicker too. Especially than using the Word Perfect software package. We used to use WordStar 2000 which had excellent function keys, we had energise/deenergise, not full, start/running, all the function keys allocated to all those sort of standard text...Since we're using Word Perfect we don't use that option, which is super useful, but here of course you just, energise, tick, deenergise, tick, and it's back to easy again. [So as far as use you think, would you say, more convenient that your current method?] My current method is purely typing. (PE2)

3.67 [How about learning, do you think the tools would be easy or difficult to learn?] It would be a hell of a lot easier to learn than Word Perfect, that's for sure. No problem with that at all.

3.68 [As far as fit to the need, do you see the tools, or the approach as a fit?] To use the tool to present that [type of FD]? [Yes.] Yes, fine. [I'd love a tool that would fit that [his style FD]. I'm biased...I think it's fine actually. Clever actually, real clever. (PE2)

3.69 [I curious about productivity or, on the other people in the plant not directly associated with FDs, like the people operating the plant or maybe some other people, what the impact of standards might be on those people.] Again, it goes back to the skill level of the operator. We've got some people out there who are just not really in

a position to pick up on what you are developing there. These are what you might say are your traditional operator who has been in the industry for quite a while and started out manually making cheese and was employed for his brawn rather than his brain, but the way we're moving is that we are trying to get much more highly qualified people in our production plants actually running a plant, but like, I think the long term goal is probably get a little bit more like the petro-chemical industry today where you've got very skilled people running the plant and it's those people who are going to take the ownership of FDs like this and that's obviously related to productivity and effort. (OE)
3.70 [In your job do you feel like a standard FD would have an effect in your job?] Well, not in my job, but from a company perspective I think what we're trying to achieve here is actually good because it means that.. ok, obviously standardising on the way we do things, which is good, it means that somebody else can come in fairly quickly pick up fairly quickly what has been done there and through the modify it or whatever, the problem we have is the Brian Robinson's or the Murray Dench's of the world aren't always going to be around...or go and work for another company and not be available for a period of time. To actually set up a standard way for somebody else can come in a pick up the pieces fairly quickly is a good idea....[You're thinking about a person who would be writing or maintaining an FD, is that what you're thinking about, or are you thinking about the PLC software guys, or both?] I'm thinking more the FD really...I suppose I'm just trying to define in my own mind what sort of person that is...perhaps ideally he is part of the existing site operational structure. Hopefully there will be more than one person who actually can and will do this on a site. But he is certainly not...has got to have a good technical basis, understand a bit about PLC writing and understand a bit about how a plant works, understand the process part of the plant. (OE)
3.71 [Your comment would relate to productivity, these people would be more productive because they could pick up this FD that follows a standard, they'd be able to understand it...?] It takes a bit of the mystique away from FD writing really to someone who is coming in fairly fresh perhaps, you got a fairly good structure there which they can work through straight away. (OE)

3.72 [You know, you've already mentioned the overall organisation I think, but that was, and I asked another question your effort and about people in the overall organisation and I think you pretty much addressed the overall organisation. What do you think?] Well, we could take another look at it. The overall goal of a company like Anchor is to improve on what we do already, as well without improvement obviously you don't keep going and this is just another means, package, I suppose of doing it. There's lots of other things that you have going on at the same time as well. Over a period of time lots of little things build up to the gains you see in a big company like Anchor. [...and so overall in Anchor I guess then if we had the standard FDs there'd be potentially more people available that can read FDs or write FDs...] Yes, I'd say so [and then if given the right skill of plant operators maybe they can be come involved in FDs...] Yes...Yes, that's and ownership sort of thing as well, where obviously if you can get the plant operator involved in writing of the FDs then they feel as though they have the ownership of the production plant... (OE)
3.73 [Do you see any drawbacks to a standard?] Drawbacks....not really....what it may need doing is, the only draw back I see, but which is not really a drawback but part of the system of controlling of FDs is that it does take a reasonable amount of

work for someone, whoever that someone is in an organisation like Anchor, to ensure that that development is ongoing. Someone needs to conscientiously pick that up as part of as part of the ... Somebody within Anchor needs to have the ownership to ensure that development is still there, it's part of somebody's job. [Ensure that the standard is evolving?] Yes...It's not necessarily a drawback, it's just something which needs to be there....[Any other comments on standards that come to mind?] No, just to say that they are encouraged really, it's something in my line of work we try to set standards as well, we're doing much the same thing as well, we constantly evolving, reissuing it, it's part of the process really. (OE)

3.74 [I'm curious about and trying to find out about was how people thought the use of these tools compared to their current methods or their current tools, whether they thought they are easier or harder to use, or easier or harder to learn, and whether they fit the need....do you have a feel on any of these, ease of use or ease of learning?] I'm probably not the right person to really comment on how it perhaps compares to what I'm doing at the moment, but the time we spent this morning working through that seemed fairly simple, fairly structured, about how you go and build your FD. Obviously somebody like myself would have to spend a bit more time to really go through it but you know, it looked fairly simple and I don't think that complex. [So you think it would be fairly easy to learn to use and...?] I would think so. [Do you think it fits the need of the...that we are trying to address...support for FDs and code generation?] Well, the proof will be in the pudding once you actually get all your links in place, but obviously that's a good start. (OE)

3.75 [Think you might feel, if you could comment down here on maintenance of FDs or code? Do you think there'd be a effect on productivity or quality as far as maintaining FDs or PLC code if we had these tools?] Yes, I think so. Again it comes back to the fact that you're generating a standard there and your quality should be maintained right across the board so much more than if you don't have a standard, different people working in it's pretty hard to actually build up a quality base there. [Do you see people being more productive also...?] I would say so. [...in the maintenance phase of the plant?] Yes. [Because of the ability to pick up a known quantity or a known, a document where they already know the structure?] Yes. (OE)

B.4 People, Structure, Tasks

4.1 ...I guess from our point of view, if we got some things done and presented in some fullness and to some standard and then we're allowed to be creative, I mean that would be the ideal. That's what I was saying before, the thing that worries us more than anything else is we don't get an advantage if we're not allowed to be creative. We really don't want to do a job, this is a bit of an outrageous statement I suppose, but if we were told exactly how to do a job, then why would be using our engineers to do that job. There are plenty of other companies can do that. We want to use our engineers creatively to give us an advantage. And if someone is telling us exactly how to do a job, then it's a pointless exercise for use, because it means that number of 20 other companies can do it exactly as well as we can, so we would want and go find a new job that nobody has every done before so that we can break into a new segment of the industry, apply a new bit of technology, or an old bit of technology in an industry that's never used it before. That's really the strategic purpose for us. And so we would try to hang onto creativity. In a business that gives us differentiation. (GM/PLC)

4.2 If this [outsourcing] continues to move this way, there's going to be two problems I would think in the industry. One is there's going to be few people who understand the process within the factories that they're running...And I guess it's a real worry to everybody, particularly that we are looking for nationally led export recovery... And so the ability to make a change anywhere in the factory is going to become less and less...who applies the process to make something work, there's a limited number of people who can do that now in NZ and that's the process/electrical engineer...I see that as being a major problem. So I'm not sure that even if we take this approach of being able to make a change to an FD and it's going to be down loaded into a control system and that therefore may eliminate these levels of expertise required, the question would still be who would have the process knowledge to understand the implications of doing that unless the rules that were based in the FD were absolutely tested.

4.3 [Reference to code generation] Get a lot more graduates working wouldn't you...Graduates, yes, cheaper people....(E2/PLC) [So standards would influence the type of people that are doing jobs? You could maybe go with a less qualified person, eventually, or a less experienced person?] Yes, less experienced. Still need someone for supervision...(E2/PLC)

4.4 I think the other thing that's happen is what we're already doing is that we've created and environment already where you're engineers are more project managers. But we have elevated over fairly brief period of time some young guys who would normally have been code writers a generation ago and we are giving them project management responsibility, they are project managers rather than....one A-B engineer to a project. (GM/PLC). [How do standards help that process?] Well, the same thing, I mean you can employ other people to come in under contract you and since it's a standard method of doing it very simple if they have worked for you on a project before that and you apply the same standard, you can be paying them project money up front, but they are going to sit down today and write it under a standard rather than under a non standard you would want to use your own people more often because of the learning process up front. You'd be paying for 6 weeks of learning. If a standard is applied, you can get a contract engineer in and say here we are, we're working to this standard and away you go and tomorrow he can be actually applying

the code. And I think that's a commercial return, and it also has a return in the way that you organise your own company. You have less engineers but giving them more responsibility in the sense that each could be running a project rather than just being a project team member. That's what we are currently trying to do isn't it...Yes, I think you could actually get a more powerful structure out of it rather than a weakened structure...But I think what it's training engineers to be managers rather than just engineers. Looking for multi-skilling. You're finding that you will have an engineer in their and he's looking after a multi-cultural group of people if you like. He's looking after OEMs, he's looking after contractors, and he's looking after product managers, production managers, and production people from the plant. And it could be quite a young engineer looking after all those people, draftsmen, contract draftsmen, contract engineers, process people, a typical group could have 1 or 2 A-B people in it, and 5 or 6 or 7 people, some from the client, some from the process OEM, some contract draft people, and what it makes him become is a manager of a team, not just a code writer. And that's significant for him in his career. If he can do those things several times well, then there's no reason why he can't go anywhere in business. That's quite different to the old days, I believe. Different experiences being given to young engineers. It's a make or break environment, because if they stuff up a project it's going to be a major stopper. But they are being launched into it much earlier in their lives...So I think there are some very powerful things come out as well. [Out of standards themselves?] Oh, absolutely, if you think about it. (GM/PLC)

4.5 From our point of view, what we want to be able to do is do it a bit smarter than that. We want to have something that, and CSE want to do that too, but I think coming back from the Anchor philosophy of having a standardised approach to everything, that will restrict them in what they want to do as well. So I think there's that other side of the thing, you know, the other side of the story, which comes back to this creativity in trying to differentiate yourself by giving a better answer rather than giving a standard answer is a conflict I would say. (GM/PLC)

4.6 [We talked about changes to work that might result from using these tools and standards, and I was asking people about whether they thought tasks would be added or removed from their jobs and how they thought relationships to other people in the effort might change. I think you've talked a little bit about the idea of engineers becoming managers. Do you have any other comments to add in that area?] I think from a business point of view what it does for us it allows for us to have a smaller more effective engineering business unit that, and allows us to employ 3rd parties outside or company to take on major projects. [That'd be standards?...] That's be standards...it would also be... are we talking about or about ....[How about the tools. That applies to standards. Does that also apply to having a CASE tool environment like this?] I think so. Once you've trained a contractor on this would you never have to train him again?] No...It's a one hit job, isn't it, I mean it's almost intuitive anyway. (GM/PLC)

4.7 [[Comment on the] tasks they [people] do changing with a standard or with tools.] Especially if the tool [FDA] is there, well just looking at what you've done so far with the pull down lists of I/O and steps I mean it is making it easy, it's basically pick and place type stuff, operators...it's not going to be so daunting for operators to change something or add something. A lot of operators if you presented them with Word Perfect and said here you go it would take them a couple of weeks to actually get used to using Word Perfect, just because it's quite a powerful package you know,

and if there's a lot of, if your FD has got a lot of control stuff and therefore fonts and indenting and all that sort of stuff, it could be a nightmare for them getting used to it. Whereas the approach there where it's a pick list and you are building it and presenting a standard formatted document at the end of it certainly that's going to take away the daunting part of it. (SM)

4.8 [... we were interested in how your work might change if you had these tools. Would you see certain tasks added or removed, or would you see your relationship to the operators or the process people changing with these kinds of tools?] Yes, I think from the PLC coding end, things would change, especially with the code generation obviously it would take away big chunk of work there, the development, producing the end product is going to be quicker. Database work and messaging is certainly going to be easier to deal with. The tools going to be easier to use than that presently exists. It's integrated into the overall, the integration thing is there. The I/O database and that, I think that there, I'd have to see it running first to see. Like once again you are competing with the databases where you can do some fairly powerful queries and stuff like that which we do. But having said that, it's obviously something that you can build on...From an operators point of view, I think what he is going to be receiving and reading will be a lot more user friendly to him, training's going be easier and I think you're going to end up that he's going to be changing FDs. And I think you're going to end up that he's going to be changing FDs I think he's going to be the guy that's going to end up being responsible for it. Which is certainly a good thing. [Do you see the software guys doing that too, maybe doing more with the FD than they did before?] Yes, as well. I think it would be a similar change. Especially at site where like the maintenance, the sparky or whatever, will be asked to go down an change something, he's going to have the tools there to change the FD straight away. Perhaps he's not just going to shunt it off to somebody else or just write it up by hand. Yes, I think the tools, it's certainly a better set of tools... (SM)

4.9 [Do you see changes in your job or their relationships with other people changing if they used these tools.] Me personally? [Yes] Yes [What do you see as changes?] Making it run.

4.10 [Do you see yourself doing more or less kinds of different things then you did before?] No, really, I'd just be one of the people, one of the administrators to make it hang together. [One question I had was might enable less experienced people to work on FDs.] Yes, online, yes, there's still, like two issues there....<end of tape, check video...> (SM2)

4.11 [A lot of what this concept does though of going the next stage from FDs to code generation, and being so detailed in the FD, is it does take away a lot of the creativity from the programmers, and they start to lose job interest, you need a different type of guy then to translate this into a program than as to what you've had in the past, they don't need to be so, self starters if you like, you don't need that person with initiative anymore because is more spoon fed. Somebody with that sort of initiative would soon get fed up with it. [So you're saying if we had the code generation then there might not be much original work as far as the PLC coding?] Yes. [And that may change the type of person you have doing that?] Yes, and also just having the FD as well being very regimented like this and almost written in PLC language if you like, well then process engineering language, it takes away a lot of that creativity, even just like that. [Actually I disagree with this [level of detail... I say

it shouldn't this say CIP202 not selected instead of this flag...] Yes. [Phil thought that it should too.] [How about if you carry that a step further, would you people that are, like let's say the good PLC people, would you see them moving maybe more into the FD writing, if they weren't doing so much the PLC coding, might they move into more involvement with FDs?] No, because it's different core skill process engineer as opposed to a programmer. Those PLC programmers would move into different levels of other computer systems which are used within the factory I think, going up into the different, more into information systems...Sort of integration and reporting type stuff. (EM)

4.12 [My last question in this category is... on changes to people's work or relationships to other people if these tools and standards were implemented. I think you mentioned the possibility that if we had automatic code generation that programming could become a less interesting task...] Yes. [Do you see any other relationships changing as far as contractors, or FD writers, or operators if these tools were available?] No, I don't think so... [Do you see other people getting involved with the development of FDs if they have the tool that provided the infrastructure?] The FD is the responsibility of the equipment supplier in most cases, unless there're automating the system themselves or if there're doing some sort of contractors to automate it, there're responsible to generate these FDs, it's not as though they'd say, you know, get some of Anchor's programmers in we can all sit down and ... don't do that... because they contracted to supply some equipment that works, that meets these specifications, it's up them to make sure it does. [OK. And the maintenance phase when the operators are using the FD as a manual, I guess that gets back this maintenance phase of if a change comes in would they keep it up to date.] That's management discipline as opposed to the tools not making that much difference. (EM)

4.13 [If we had this environment, can you see changes to your work, like tasks added or removed from what you do or maybe how relationships would change with other people like the FD writer or operators] In some ways it might sort of limit or reduce the workload of the PLC coder but then especially if everything was related back to what was happening within the FD mainly that was generated from the FD then it would probably reduce the amount of work for sure. [Reduce the coding?] Yes, but also there could be more work maybe in writing the FDs but it shouldn't really be, it should be less work. [Do you think maybe you would be modifying FDs to get the code to..] I think yes, it probably would happen maybe if you were given an FD from somewhere on this or whatever and you went through, maybe it's a second stage where you can see that things might not work as written or maybe you could add the extra stage where you are adding in extra information for the operator so it could be sort of a more broader base than going into that sort of thing. [So you are saying maybe you need something in the FD for the MMI or the operator that the FD writer didn't put in there and you could see yourself changing the FD to...] Yes, to increase the amount of information... Yes, I think as far as information, there would need to be more overlapping of what's involved. The FDA will insure that it's in a structure so the FD writer doesn't need to know what's going on because so long as it's confirming to the structure it's then going to happen but there's got to be ... tend to spread out more. [So you are saying maybe the line between what the process engineer does and what the software engineer does, maybe that line is going to blur a little more, maybe there will be more overlap.] Yes, that would be a benefit I think. (SE)

4.14 [Given a choice would you prefer the new or the current methods?] I think as far as an interesting job sort of thing, you'd want to prefer the old because there's much more diversity in what you are doing so it's more interesting going on. I think if I was working with that tool for 3 years, I'd be looking for other things to do. [Because of the code generation?] Yes, I mean because there'd be less individuality in what's going on but there's plenty of things out there that you can sort of do from that so I don't think there's so much discipline, but it's probably more of a stage where you'd write code for a little while using that method and then you'd want to go and do something else. [Maybe you'd move up to writing the FD?] Yes, sort of tend to be more cross-skilled, that sort of thing. [So as far as the environment itself, your main objection would be the interest in the job, not that the tools are going to get in your way?] Yes, well they get in your way as far as what you are limited to is what the guy who wrote the FD tells you to do so you've still got to write the code like that, but then there's the ability to do things a wee bit differently maybe a bit more powerfully or things like that, but yes for me it would be more of an interest sort of thing after you've done a project go and start off and knew the structure then it tends to be more of a typists job than a programmers job. [Yes, you might be more involved in configuring the addresses and..] Yes, of the system... (SE)
4.15 [Would there be any reason why you would want to switch to the new methods?] Well, personally there wouldn't be....yes, I mean it would probably cut down in errors as far as typos, things like that, checking that the switch is energised/deenergised, if it's done automatically then it's going to be there, it's not a possible error like, a lot of the work is fairly mundane, especially fault checking and things like that and it would get rid of those tedious sort of tasks. [So it might get rid of the tedious tasks, but also some of the more interesting parts too. Is that what you're thinking?] Yes the tedious tasks because of the structure, and the more interesting parts also because of the structure because you couldn't change what was going on. So I think it would, the tedious parts are a lot more work than the sort of different parts. The benefits would out-weight the others... (SE)

4.16 [Next question: if you think about the tasks that you do in your work, do you see changes in what you do in your work, do you see tasks added or removed from your job?] Well I would hope that certain tasks would be removed, like the, just the development and maintenance of the data base is one of the biggest thing. Ideally I can get a disk of the I/O and import it and that's it. That's the type of thing. [Would you see yourself doing more with the FD since you are doing less with the code...?] Yes, we'd would probably be developing FDs ourselves, rather than just coding off them and to testing and simulation stage... Be involved in that I suppose. (SE)
4.17 [Do you see the process engineer doing less on the FD and the software guy maybe doing more? Like maybe putting the changes himself into the FD?] Yes, definitely. And you could have process people themselves, senior production type people updating their own FDs. [When you say the process people, that's the guys making the product out in the factory?] Yes. [So they may make changes to FDs as well as you the software guy if you have this environment?] Yes, they could get more involved in the project. (SE)
4.18 [That gets down to here, relationships with other people in the factory or the whole organisation. Do you see the tools changing relationships, like the process people maybe getting more involved in the FD, with the software guy, with the engineer?] Yes, I think so, I would try to involve them as much as possible, but it

allows them to become more involved, because the system, once they learn it and they know the fundamentals of the plant, the can be involved in productive rather than turning out something that is not useable. [So you are saying the fact that there is a standard that they understand and then there's the tools that are easy to use would allow them to get more involved with maintaining the FD or developing the FD?] Developing and maintaining... Usually on each site there's a few people who get involved in the process development and design and it would give them the tools without having to give their ideas to someone else. [So you are saying they may become more productive because they can interact more directly with the FD without having to go through intermediate people?] Yes, they can work away on it and bring it to us to look at when they've got something useful, rather than a document that's just hand written because they don't know how to use a word processor probably or what ever, and we can sit down and change it together if you want, I'd be more productive. (SE)

4.19 [Comments about the programmers -- if there was a standard?] ...you'd end up with a situation where at the moment you've got people who have ended up specialising in one facet this, you've got 2 or 3 specialists, really what you are doing is cutting down the number of specialty functions, that can only be an improvement. You'll find that those people who have spent their time learning rules, being an expert and learning rules of coding or being and expert and learning rules to do this FD, those jobs become on longer quite as important and the time you've spent in becoming an expert on that you become an expert in thinking about different things that are more productive in terms of the overall result. That's where I've thought all along, that's why I'm enthusiastic about the whole thing, the standardisation. (PE)
4.20 [If you had [simulation] capability would it improve your productivity or quality or some other measure of effectiveness?] I hadn't thought before about using that sort of system, you are throwing the subject up today, and I hadn't really thought about having the tool that would enable me to do a bit of simulation before. It would have to be a plus but it would depend on long they give you, at the moment on most jobs the number of people doing that, the time that's available to do it, you probably don't have a lot of time to do it, given sufficient time, a bit of extra time, a tool that is sufficiently easy to use, you might get right into it. (PE)
4.21 [Can you see any changes to the tasks you perform or your relationships with others in the organisation if you used these tools?] The essence of what I do would stay exactly the same. But the relationships would change because in the way I was talking about before if you end up with a whole job function which is really not there, those people would tend to do this job or a mixture I suppose...There'd be more people doing this...[There'd be more people doing the FD or feeding information back to you?] Hmmm...It may become simpler, you may be able, my role as FD writer may end up doing the process description I suppose. The person who's familiar with the process may not need to get as deeply into the logic as I have and completely define the logic. It may be that the process engineer like myself may really describe the process in more general terms and people who now have experience in programming and coding will start generating FDs from a different ... a lot of programming in NZ is already done that way ... you end up with process description and you get people specialising in programming without process background taking it from there but I think at the moment, without these tools that's not a good way to separate out the

project. It depends on the process capabilities of the programmers. So there could be better and more productive ways of using people's expertise. (PE)

4.22 [If we had this set of tools and we had this environment where we could generate some code, do you see any changes to the way you would do your work in terms of would you see new tasks added to your work or tasks shifted to someone else, or changes in the way you work with other people involved in the development?] Tasks, I would imagine would remain pretty much the same. I can't...no I can't see a major change there. And relations to others, same deal, I don't see that changing there. Mark me either plus or minus. No, no, nothing there on that one. (PE)

4.23 [Rather than productivity, do you see effects on quality or any other kind of measures?] Well, one would certainly hope so...one of the problems we have is, I probably mentioned this already, because there are so few really good FD writers, those people are, during our busy season, fairly stretched working on various contracts. Of course what tends to happen with us is that we decide to do another project fairly later on in the process, so those Brian Robinsons, those Murray Denches are not available to actually do the FD writing or do the PLC code work. Having a structure like this means, OK, we can get somebody in there who may not necessarily be Brian Robinson or Murray Denche, but he's going to get the structures and the basics right by having a form to work with, and if necessary perhaps we can get, we can do some sort of audit on what they've done by a Brian Robinson or a Murray Dench equivalent when they are more readily available to do a fine tune or a post FD audit, post FD commission audit on it. I suppose what I'm looking at is just making more people available to do this work. Commissioning has been done by far fewer. That all relates, OK, it's productivity and quality thing that you're actually going to get the plant up and running...But, quality would also come back to the audits really. There has to be some mechanism where you can, where the standard is actually audited. There is a standard maintained through out. (OE)

4.24 [...So you're saying that having a standard gives us a pool of more people that create FDs. In testing of the plant, like commissioning and maintenance, do you see any other implications of standards, or pretty much the same comments in those phases also?] Yes, probably the same. Like anything else it's really just building up you skill base as well. (OE)

4.25 [This is what I mentioned to you earlier, I was asking people if they saw any changes to their work as a result of using these tools and standards. Whether they saw tasks added or removed, or relationships to others changing. As far as the organisation is concerned, would you see relationships changing to any of these people involved, or tasks shifting around?] Yes, I think I probably touched on it before, I made the comment that there are only so many Brian Robinsons and Murray Denches out there, one would hope that they would see that the development of a standard FD is going to make their own work easier and they can possibly go the next step up the trade, whatever that is for them. [That would be the standards, they do that. How about the tools themselves, would they also figure into that? Having the FDA and maybe even the code generator?] I suppose what it does is it, because we (Anchor) would now be the enforcer of the standards... (OE)

Bibliography

Abbott, 1993. Abbott, B. and Bapty, J. Model-Based Software Synthesis. *IEEE Software 10*, 3 (May 1993), 42-52.

Ahlgrim, 1989. Ahlgrim, S.C. and Chang, T.C. "A Survey on the Use and Development of Computer Aided Process Planning Systems" Technical Report TR-ERC 89-6, Purdue University, West Lafayette, IN, USA, April 1989.

Ahmed, 1988. Ahmed, K. *Development of a computer software for the design of sequential logic circuits (ladder diagram)*, Master's thesis, University of Illinois at Chicago, 1988.

Anchor, 1993. *New Zealand Dairy Group of Companies Annual Report 1993*, Hamilton, New Zealand.

Antill, 1985. Antill, L. Selection of a Research Method. In *Research Methods in Information Systems*. North-Holland, Mumford, E., Hirschheim, R., Fitzgerald, G., and Wood-Harper, T., pp. 203-215, Amsterdam, The Netherlands, 1985.

Arango, 1991. Arango, G. and Prieto-Diaz, R. *Domain Analysis and Software Systems Modeling,* IEEE Computer Society Press, Los Alamitos, CA (1991).

Argyris, 1985. Argyris, C., Putnam, R., and Smith, D.M. *Action Science,* Jossey-Bass Inc., San Francisco, CA , Jossey-Bass management Series(1985).

Ayres, 1989. Ayres, R.U. Technology Forecast for CIM. *Manufacturing Review 2*, 1 (March 1989), 43-52.

Barb, 1987. Barb, M.E. Terminology and a Structural Model for Batch Control Systems. In *Proceedings of the Sixth Annual Control Engineering Conference,* Rosemont, Illinois, 1987, pp. 62-66.

Barstow, 1985. Barstow, D.R. Domain-Specific Automatic Programming. *IEEE Transactions on Software Engineering SE-11*, 11 (November 1985), 1321-1336.

Barstow, 1991. Barstow, D. Automatic Programming for Device-Control Software. In *Automating Software Design*. AAAI Press/MIT Press, Lowry, M.R. and McCartney, R.D., pp. 123-140, 1991.

Basili, 1988. Basili, V.R.B. and Rombach, H.D. The TAME Project: Towards Improvement-Oriented Software Environments. *IEEE Transactions on Software Engineering 14*, 6 (June 1988), 758-772.

Baxter, 1992. Baxter, I.D. Design Maintenance Systems. *CACM 35*, 4 (April 1992), 73-89.

Beck, 1986. Beck, C.L. and Krogh, B.H. Models for Simulation and Discrete Control of Manufacturing Systems. In *Proceedings IEEE International Conference on Robotics and Automation,* IEEE Computer Society, IEEE Computer Society Press, Washington, DC, April 1986, pp. 305-310.

Benbasat, 1984. Benbasat, I. An Analysis of Research Methodologies. In *The Information Systems Research Challenge...Proceedings*. Harvard Business School Press, McFarlan, F.W., pp. 47-85, Boston, Mass, 1984.

Benbasat, 1987. Benbasat, I., Goldstein, D.K., and Mead, M. The Case Research Strategy in Studies of Information Systems. *MIS Quarterly* (September 1987), 369-386.

Bennis, 1985. Bennis, W.G., Benne, K.D., and Chin, R. *The Planning of Change,* Holt, Rinehart and Winston, New York , Fourth(1985).

Bhatnagar, 1990. Bhatnagar, S. and Linn, R.J. Automatic Programmable Logic Controller Program Generator With Standard Interface. *Manufacturing Review 3*, 2 (1990), 98-105.

Bristol, 1985. Bristol, E.H. A Design Tool Kit for Batch Process Control: Terminology and a Structural Model. *InTech 32*, 10 (1985), 47-50.

Brown, 1994. Brown, A.W., Carney, D.J., Morris, E.J., Smith, D.B., and Zarrella, P.F. *Principles of CASE Tool Integration,* Oxford University Press, New York, N.Y. (1994).

Bruyn, 1988. Bruyn, W., Jenson, R., Keskar, D., and Ward, P. ESML: An Extended Systems Modeling Language. *ACM Software Engineering Notes 13*, 1 (Jaunary 1988), 58-67.

Bunch, 1994. Bunch, M. and Love, J. Towards a Standard for Batch Process Control. *Measurement + Control 27*, 3 (April 1994), 74-77.

Busby, 1992. Busby, J.S. and Hutchinson, D. The Practical Integration of Manufacturing Applications. *Software-Practice and Experience 22*, 2 (February 1992), 183-207.

Buxton, 1980. Buxton, J.N., "Requirements for Ada Programming Support Environments (APSEs) -- Stoneman" , US Department of Defense, Washington, D.C., USA (1980).

Card, 1987. Card, D.N., McGarry, F.E., and Page, G.T. Evaluating Software Engineering Technologies. *IEEE Transactions on Software Engineering 13*, 7 (July 1987), 845-851.

Cassell, 1983. Cassell, D.A. *Microcomputers and Modern Control Engineering,* Reston Publishing Company, Inc., Reston Virginia (1983).

Chaar, 1990. Chaar, J.K. *A Methodology for Developing Real-Time Control Software for Efficient and Dependable Manufacturing Systems*, Ph.D. dissertation, University of Michigan, Ann Arbor, MI, USA (1990).

Chaar, 1991. Chaar, J.K., Volz, R.A., and Davidson, E.S. An Integrated Approach to Developing Manufacturing Control Software. In *Proceedings of the 1991 IEEE International Conference on Robotics and Automation,* IEEE, IEEE Press, Sacramento, California, April 1991, pp. 1979-1984.

Checkland, 1990. Checkland, P. and Scholes, J. *Soft Systems Methodology in Action,* John Wiley, Chichester (1990).

Checkland, 1991. Checkland, P. From Framework through Experience to Learning: the Essential Nature of Action Research. In *Information Systems Research: Contemporary Approaches and Emergent Traditions*. Elsevie Science Publishers B.V., Nissen, H.E., Klein, H.K., and Hirschheim, R., pp. 397-403, 1991.

Chisholm, 1993. Chisholm, R.F.C. and Elden, M. Features of Emerging Action Research. *Human Relations 46*, 2 (February 1993), 275-298.

Cleaveland, 1988. Cleaveland, J.C. Building Application Generators. *IEEE Software 5*, 4 (July 1988), 25-33.

Dart, 1987. Dart, S.A. Software Development Environments. *IEEE Computer 20*, 11 (November 1987), 18-28.

David, 1992. David, R. and Alla, H. *Petri Nets & Grafcet Tools for modelling discrete event systems,* Prentice Hall International, Hertfordshire, UK (1992).

Davis, 1989. Davis, F.D. Perceived Usefulness, Perceived Ease of Use, and User Acceptance of Information Technology. *MIS Quarterly 13*, 3 (September 1989), 319-339.

DeLone, 1992. DeLone, W.H. and McLean, E.R. Information Systems Success: The Quest for the Dependent Variable. *Information Systems Research 3*, 1 (March 1992), 60-95.

DeMarco, 1978. DeMarco, T. *Structured Analysis and System Specification,* Yourdon Inc. (1978).

Devanathan, 1991. Devanathan, R. Computer Aided Design of Sequential Control System Using State Transition Diagram and AI Techniques. *Journal of Electrical and Electronics Engineering, Australia 11*, 4 (December 1991), 227-232.

Ekberg, 1987. Ekberg, G. and Krogh, B.H., "Prototype Software for Automatic Gneeration of On-line Control Programs for Discrete Manufacturing Processes" Technical Report, Pittsburg, Pennsylvania 15213, February 1987.

Elden, 1993. Elden, M. and Chisholm, R.F.C. Emerging Varieties of Action Research: Introduction to the Special Issue. *Human Relations 46*, 2 (February 1993), 121-141.

Falcione, 1993. Falcione, A. and Krogh, B.H. Design Recovery for Relay Ladder Logic. *IEEE Control Systems* (April 1993), 90-98.

Fisher, 1990. Fisher, T.G. *Batch Control Systems Design, Application, and Implementation,* Instrument Society of America, Research Triangle Park, North Carolina (1990).

Foster, 1972. Foster, M. An Introduction to the Theory and Practice of Action Research in Work Organizations. *Human Relations 25*, 6 (1972), 529-556.

Frenkel, 1985. Frenkel, K.A. Toward Automating the Software-Development Cycle. *Communications of the ACM 28*, 6 (June 1985), 578-589.

Fuggetta, 1993. Fuggetta, A. and di Milano, P. A Classification of CASE Technology. *IEEE Computer 26*, 12 (December 1993), 25-38.

Galliers, 1992. Galliers, R. Choosing Information Systems Research Approaches. In *Information Systems Research*. Blackwell Scientific Publications, Galliers, R., Ch. 8, pp. 144-162, Oxford, England, 1992.

Gayman, 1988. Gayman, D.J. An Old Favorite Gets New Standards. *Manufacturing Engineering* (January 1988), 55-58.

Gee, 1992. Gee, D.M., Worrall, B.P., and Henderson, W.D. Real-time CASE Tools: A Review of Current Tools and Future Prospects. In *CASE Current Practice Future Prospects*. John Wiley & Sons Ltd, Spurr, K. and Layzell, P., Ch. 5, pp. 67-83, West Sussex, England, 1992.

Gibbs, 1994. Gibbs, W.W. Software's Chronic Crisis. *Scientific American 271*, 3 (September 1994), 86-95.

Gibson, 1975. Gibson, C.F. A Methodology for Implementation Research. In *Implementing Operations Research/Management Science*. Elsevier Publishing Company, Schultz, R.L. and Selvin, D.P., Ch. 4, pp. 53-73, 1975.

Grady, 1987. Grady, R.B. Measuring and Managing Software Maintenance. *IEEE Software* (September 1987), 35-45.

Gunasena, 1991. Gunasena, N.U. and Lehtihet, E.A. Automatic Program Generation for PC-Based Sequence Control Problems. *Journal of Manufacturing Systems 10*, 2 (1991), 109-120.

Habib, 1990. Habib, A. Sequence Matrix Simplifies Software Documentation. *Intech* (July 1990), 38-41.

Halang, 1990. Halang, W.A. and Stoyenko, A.D. Comparative Evaluation of High-Level Real-Time Programming Languages. *Real-Time Systems 2*, 4 (1990), 365-382.

Halang, 1992. Halang, W.A. and Sacha, K.M. *Real-Time Systems,* World Scientific Publishing Company, Singapore (1992).

Hamilton, 1992. Hamilton, S. and Ives, B. MIS Research Strategies. In *Information Systems Research*. Blackwell Scientific Publications, Galliers, R., Ch. 7, pp. 132-143, Oxford, England, 1992.

Harel, 1990. Harel, D., Lachover, H., and A. Naamad, STATEMATE: A Working Environment for the Development of Complex Reactive Systems. *IEEE Transactions on Software Engineering 16*, 4 (April 1990), 403-414, also in Kavi, Krishna M., Real Time Systems, abstractions, languages and design methodologies, IEEE Computer Society Press, Los Alamitos, CA, 1992, pp. 609-620.

Harel, 1992. Harel, D. Biting the Silver Bullet Toward a Brighter Future for System Development. *IEEE Computer 25*, 1 (January 1992), 8-20.

Hatley, 1987. Hatley, D.J. and Pirbhai, I.A. *Strategies for Real-Time System Specification,* Dorset House , New York, NY (1987).

Henderson, 1990. Henderson, J.C. and Cooprider, J.G. Dimensions of I/S Planning and Design Aids: A Functional Model of CASE Technology. *Information Systems Research 1*, 3 (September 1990), 227-254.

Herget, 1988. Herget, C.J. Survey of existing computer-aided design in control systems packages in the USA. In *Proceedings of the 4th IFAC International Symposium,* CACSD□'88, Beijing, 1988, pp. 46-48.

Higginson, 1990. Higginson, J. PLC Software Moves into the New Decade. *Control & Instrumentation 22*, 4 (April 1990), 90-91.

IEC, 1988. IEC. *Preparation of function charts for control systems,* Publication 848, 1988, (Bureau Central de la Commission Electrotechnique Internationale; 3 rue de Varembe; Geneve, Suisse), .

IEC, 1991. IEC. *Draft IEC 1131-1: Programmable controllers - Part 3: Programming languages,* Draft Standard, 1991, (Bureau Central de la Commission Electrotechnique Internationale; 3 rue de Varembe; Geneve, Suisse), .

Interactive, 1991. *Software through Pictures: Products & Services Overview,* Interactive Development Environments, San Francisco, CA, USA, 1991.

Jamshidi, 1985. Jamshidi, M. and Herget, C.T. *Computer-aided Control Systems Engineering,* North-Holland, New York, NY USA (1985).

Jenkins, 1985. Jenkins, A.M. Research Methodologies and MIS Research. In *Research Methodologies in Information Systems.* North-Holland, Mumford, E., Hirschheim, R., Fitzgerald, G., and Wood-Harper, T., pp. 103-117, Amsterdam, The Netherlands, 1985.

Jewell, 1991. Jewell, M.T. Integrating People and Technology Through Object Oriented Graphics Programming. In *IPC '91 Proceedings of the 20th Annual ESD International Programmable Controllers Conference and Exposition,* ESD-The Engineering Society, ESD, Detroit, Michigan, April 1991, pp. 601-612.

Johnson, 1987. Johnson, D.G. *Programmable Controllers for Factory Automation,* Marcel Dekker, Inc. (1987).

Jonsson, 1991. Jonsson, S. Action Research. In *Information Systems Research: Contemporary Approaches and Emergent Traditions.* North-Holland, Nissen, H.E., Klein, H.K., and Hirschheim, R., pp. 371-395, Amsterdam, The Netherlands, 1991.

Kavi, 1992. Kavi, K.M. and Yang, S.M. Real-Time Systems Design Methodologies: An Introduction and a Survey. In *Real-Time Systems Abstractions, Languages, and Design Methodologies*. IEEE Computer Society Press, Kavi, K.M., Ch. 4, pp. 539-553, Los Alamitos, CA, 1992.

Keebler, 1993. Keebler, F. *An Executable Specifications Approach to Component-Oriented Software Development*, Ph.D. dissertation, Wayne State University, Detroit, Michigan, 1993.

Kemerer, 1989. Kemerer, C.F. An Agenda for Research in the Managerial Evaluation of Computer-Aided Software Engineering (CASE) Tool Impacts. In *Proceedings of the 22nd Hawaii International Conference on System Sciences,* Kailua, Hawaii, 1989, pp. 219-228.

Kopetz, 1991. Kopetz, H. The design of real-time systems: from specification to implementation and verification. *Software Engineering Journal* (May 1991), 72-83.

Krogh, 1987a. Krogh, B.H., Willson, R., and Pathak, D. Automatic Generation of Control Programs for Discrete Manufacturing Processes. *Robotics Institute, Carnegie-Mellon University, 1987 Annual Research Review* (1987), 21-31.

Krogh, 1987b. Krogh, B.H. and Ekberg, G. Automatic Programming of Controllers for Discrete Manufacturing Processes. In *Proceedings International Federation of Automatic Control 10th Triennial World Congress,* International Federation of Automatic Control, Munich, Germany, 1987, pp. 161-165.

Krogh, 1988. Krogh, B.H., Willson, R., and Pathak, D. Automated Generation and Evaluation of Control Programs for Discrete Manufacturing Processes. In *Proceedings 1988 International Conference on Computer Integrated Manufacturing,* Rensselaer Polytechnic Institute, IEEE Computer Society, Los Alamitos, CA, May 1988, pp. 92-99.

Krueger, 1992. Krueger, C.W. Software Reuse. *ACM Computing Surveys 24*, 2 (June 1992), 131-183.

Kumar, 1992. Kumar, K. and Welke, R.J. Methodology Engineering: A Proposal for Situation-Specific Methodology Construction. In *Challenges and Strategies for Research in Systems Development*. John Wiley & Sons, Cotterman, W.W. and Senn, J.A., Ch. 15, pp. 257-269, 1992.

Kurki-Suonio, 1994. Kurki-Suonio, R. Real Time: Further Misconceptions (or Half-Truths). *IEEE Computer 27*, 6 (June 1994), 71-76.

Kyng, 1991. Kyng, M. Cooperative Design: Bringing Together the Practices of Users and Designers. In *Information Systems Research: Contemporary Approaches and Emergent Traditions*. Elsevier Science Publishers B.V., Nissen, H.E., Klein, H.K., and Hirschheim, R., pp. 405-415, Amsterdam, The Netherlands, 1991.

Larson, 1985. Larson, S.A. *An Integrated Approach To Software Design For Programmable Controller Systems*, Master's thesis, University of Tennessee, Knoxville, Tennessee, June 1985.

Leavitt, 1965. Leavitt, H.J. Applied Organisational Change in Industry: Structural, Technological and Humanistic Approaches. In *Handbook of Organisations*. Rand McNally, March, J.C., Chicago, 1965.

Lempp, 1988. Lempp, P. and Lauber, R. What Productivity Increases to Expect from a CASE Environment: Results of a User Survey. In *Productivity: Progress, Prospects, and Payoff, Proceedings of the 27th Annual Technical Symposium,* Gaithersburg, MD, June 1988, pp. 13-19.

Levy, 1986. Levy, L.S. A Metaprogramming Method and its Economic Justification. *IEEE Transactions on Software Engineering SE-12*, 2 (February 1986), 272-277.

Lowry, 1991. *Automating Software Design,* AAAI Press/The MIT Press (1991).

Ludwig, 1991. Ludwig, L.G. *Evaluation and Application of a Computer Aided Software Engineering Tool for Real-Time Design*, Master's thesis, State University of New York, Binghamton, 1991.

Marciniak, 1994. *Encyclopedia of Software Engineering,* John Wiley & Sons, Inc., New York, NY , Vol. 1(1994).

McCartney, 1991. McCartney, R.D. Knowledge-Based Software Engineering: Where We Are and Where We Are Going. In *Automating Software Design*. AAAI Press/MIT Press, Lowry, M.R. and McCartney, R.D., pp. xii-xxxi, 1991.

McClure, 1988. McClure, C. The CASE for Structured Development. *PC Technical Journal 6*, 8 (August 1988), 51-67.

McClure, 1989. McClure, C. *CASE is Software Automation,* Prentice Hall, Englewood Cliffs, New Jersey (1989).

McDermid, 1991. *Software Engineer's Reference Book,* Butterworth Heinemann, Oxford, England (1991).

McLendon, 1992. McLendon, J. *Case Studies in Computer Aided Software Engineering: A Strategy for Successful Implementation*, Master's thesis, University of South Carolina, 1992.

Mettala, 1992. Mettala, E. and Graham, M.H., "The Domain-Specific Software Architecture Program" Special Report CMU/SEI-92-SR-9 Carnegie Mellon University, Pittsburg, PA, June 1992.

Munro, 1992. Munro, N. Development of Computer-Aided Control System Design Tools. *Measurement + Control 25*, 5 (June 1992), 134-136.

Myers, 1990. Myers, B.A. Taxonomies of Visual Programming and Program Visualisation. *Journal of Visual Languages and Computing 1*, 1 (March 1990), 97 - 123, MP-92-02.

Neidert, 1991. Neidert, R. The Object Oriented Paradigm and Industrial Control. In *IPC '91 Proceedings of the 20th Annual ESD International Programmable Controllers Conference and Exposition,* ESD-The Engineering Society, ESD, Detroit, Michigan, April 1991, pp. 495-505.

Neighbors, 1984. Neighbors, J.M. The Draco Approach to Constructing Software from Reusable Components. *IEEE Transactions on Software Engineering SE-10*, 5 (September 1984), 564-574.

Norman, 1989. Norman, R.J. and Nunamaker, J.F. CASE Productivity Perceptions of Software Engineering Professionals. *Communications of the ACM 32*, 9 (September 1989), 1102-1108.

Orlikowski, 1989. Orlikowski, W.J. Division Among the Ranks: The Social Implications of CASE Tools for System Developers. In *Proceedings of the Tenth International Conference on Information Systems,* Society for Information Management, ACM, Boston, Mass, December 1989, pp. 199-210.

Pathak, 1989. Pathak, D.K. and Krogh, B.H. Concurrent Operation Specification Language, COSL, for Low-Level Manufacturing Control. *Computers in Industry 12*, 2 (May 1989), 107-122.

Potts, 1993. Potts, C. Software-Engineering Research Revisited. *IEEE Software 10,* 5 (September 1993), 19-28.

Ready, 1991. *CARDtools Product Brief,* Ready Systems, Sunnyvale, CA, USA, 1991.

Rich, 1988. Rich, C. and Waters, R.C. Automatic Programming: Myths and Prospects. *IEEE Computer* (August 1988), 40-51.

Roberts, 1990. Roberts, C.A. *Generation of Relay Ladder Logic From A Real-Time Network Control Specification,* Ph.D. dissertation, Virginia Polytechnic Institute & State Univ., May 1990.

Schultz, 1975. Schultz, R.L. and Slevin, D.P. Implementation and Organizational Validity: An Empirical Investigation. In *Implementing Operations Research/Management Science.* American Elsevier, Schultz, R.L. and Slevin, D.P., pp. 153-182, 1975.

Sciences, 1989. QED Sciences (author not cited). *CASE The Potential and the Pitfalls,* QED Information Sciences, Inc., Wellesley, MA (1989).

Shaw, 1982. Shaw, W.T. *Computer Control of Batch Processes,* EMC Controls, Inc., Cockeysville, MD (1982).

Sommerville, 1989. Sommerville, I. *Software Engineering,* Addison-Wesley Publishing Company, Reading, MA , 3rd(1989).

Stear, 1992. Stear, E.B. and Shepherd, J.T., "Automated Software Generation Approaches for the Design and Development of Guidance and Control Systems Software" Advisory Report, Neuilly Sur Seine, France, March 1992.

Stewart, 1982. Stewart, C. and William Cash, J. *Interviewing principles and practices,* Wm. C. Brown Company, Dubuque, Iowa , third(1982).

Susman, 1978. Susman, I. and Evered, D. An Assessment of the Scientific Merits of Action Research. *Administrative Science Quarterly 23*(1978), 582-603.

Troy, 1987. Troy, D. An Evaluation of CASE Tools. In *Proceedings IEEE Computer Software and Applications (COMPSAC) Conference,* IEEE Computer Society, IEEE Computer Society, Tokyo, Japan, 1987, pp. 124-130.

Troy, 1992. Troy, D.A., Nugehally, N., Farooq, S., and Hergert, D. Object-Oriented Flexible Manufacturing at Miami University. In *International Conference on Object Oriented Manufacturing Systems,* University□of Calgary, University of Calgary, May 1992, pp. 358-363.

VanDerBoom, 1988. VanDerBoom, A.J.W. CACSD Developments in Europe. In *Proceedings of the 4th IFAC International Symposium,* CACSD '88, Beijing, 1988, pp. 51-59.

Vessey, 1992. Vessey, I., Jarvenpaa, S.L., and Tractinsky, N. Evaluation of Vendor Products: CASE Tools as Methodology Companions. *CACM 35,* 4 (April 1992), 90-104.

Viswanadham, 1987. Viswanadham, N. and Narahari, Y. Coloured Petri Net Models for Automated Manufacturing Systems. In *Proceedings of the 1987 IEEE International Conference on Robotics & Automation,* IEEE, IEEE, March 1987, pp. 1985-1987.

Wallace, 1987. Wallace, R., Stockenberg, J., and Charette, R. *A Unified Methodology for Developing Systems,* McGraw-Hill (1987).

Ward, 1985. Ward, P.T. and Mellor, S.J. *Structured Development for Real-Time Systems, Vols. 1-3,* Yourdon Press, New York, NY (1985).

Ward, 1986. Ward, P. The Transformation Schema: An Extension of the Data Flow Diagram to Represent Control and Timing. *IEEE Transactions on Software Engineering 12*, 2 (February 1986), 128-210.

Ward, 1989. Ward, M., Calliss, F.W., and Munro, M. The Maintainer's Assistant. In *Proceedings Conference on Software Maintenance--1989,* IEEE Computer Society, IEEE Computer Society Press, Washington, D.C., 1989, pp. 307-315.

Waters, 1985. Waters, R.C. The Programmer's Apprentice: A Session with KBEmacs. *IEEE Transactions on Software Engineering SE-11*, 11 (November 1985), 1296-1320.

Weiderman, 1986. Weiderman, N.H., Habermann, A.N., Borger, M.W., and Klein, M.H. A Methodology for Evaluating Environments. In *Proceedings of the ACM SIGSOFT/SIGPLAN Software Engineering Symposium on Practical Software Development Environ,* Association for Computing Machinery, Association for Computing Machinery, New York, N.Y., December 1986, pp. 199-207.

Winograd, 1986. Winograd, T. and Flores, F. *Understanding Computers and Cognition: A New Foundation for Design,* Ablex Publishing Corporation, Norwood, New Jersey (1986).

Wood, 1989. Wood, D.P. and Wood, W.G., "Comparative Evaluations of Four Specification Methods for Real-Time Systems," Technical Report CMU/SEI-89-TR-36 ESD-89-TR-47, Carnegie Mellon University, Pittsburg, PA, December, 1989.

Wood-Harper, 1985. Wood-Harper, T. Research Methods in Information Systems: Using Action Research. In *Research Methods in Information Systems.* North-Holland, Mumford, E., Hirschheim, R., Fitzgerald, G., and Wood-Harper, T., pp. 169-191, Amsterdam, The Netherlands, 1985.

Wynekoop, 1991. Wynekoop, J.L. and Conger, S.A. A Review of Computer Aided Software Engineering Research Methods. In *Information Systems Research: Contemporary Approaches & Emergent Traditions.* North-Holland, Nissen, H.E., Klein, H.K., and Hirschheim, R., pp. 301-325, Amsterdam, The Netherlands, 1991.

Yin, 1989. Yin, R.K. *Case Study Research Design and Methods,* SAGE Publications (1989).

Zhou, 1990. Zhou, M.C. and DiCesare, F. A Petri Net Design Method for Automated Manufacturing Systems with Shared Resources. In *Proceedings of the 1990 IEEE International Conference on Robotics & Automation,* IEEE, IEEE Press, May 1990, pp. 526-531.