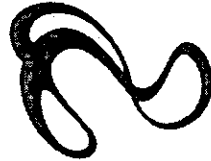


MX 0700407 9



**Middlesex  
University**

**An investigation of the utility and value of process patterns in the  
management of software development projects**

**Ahmad Hajizadeh Estabraghy**

**Thesis Submitted for the Degree of Doctor of Philosophy  
At Middlesex University**

**School of Computing Science  
Middlesex University  
London**

**June 2008**

**Director of Studies: Prof. Darren Dalcher  
Supervisors : Prof. Colin Tully  
: Prof. Anthony White**

To understand is to perceive patterns

*Isaiah Berlin*

## Acknowledgements

I would like to offer my sincere thanks to the following people for their help and support in this project.

First of all to my Director of Studies, Professor Darren Dalcher and my supervisors Professor Colin Tully and Anthony White for their guidance, encouragement and support throughout the life of this research project. Their support was in particular essential in getting approvals from various parties, in particular the Ethics Committee, for the experimental research method in this study to go ahead. As module leader for the experimented modules, Professor White played a significant part in allowing his modules to take part in this unprecedented (at Middlesex University) experimental study, and in the strategic and ethical issues concerned with the experiment. Without his interest, help and support, this research work would not have been possible.

Thanks also to Dr. Dan Diaper for giving me valuable advice and insights in the design of the experiment for this study.

I would also like to thank (CMT3991 and CMT3992) module coordinators Dr Mike Censlive, John Platts and Dr Yuan Luo (at the three campuses TP, HE and TM respectively) for their support. I would, in particular, like to thank Dr Mike Censlive for his help and support in conducting the experiment. Dr Censlive generously offered me his extensive experiences and knowledge of the details and structures of CMT3991 and CMT3992 modules and gave me valuable suggestions for the experiment design through many meetings.

I would also like to thank Elaine Sheehan, school secretary at TM, for her help in various clerical tasks with the experimentation.

Finally, I would also like to thank all the CMT3991 and CMT3992 tutors and supervisors for their support of this study and for allowing me to attend their seminars and talk to their students.

P1269052

Site HE	MIDDLESEX UNIVERSITY LIBRARY
Accession No.	0700407A
Class No.	005.1 EST
Special Collection ✓	

---

## Abstract

Pattern theory has engendered much controversy in the field of architecture; yet it has brought new insights to the field of software engineering. Patterns continue to play an important role in software engineering in general, and in software development in particular. In this study, two preliminary surveys, focusing on the two fields of architecture and software engineering, were carried out to investigate the role and effect of patterns. The surveys indicate that while, patterns are unpopular within the architecture community and are criticised for stifling creativity, software patterns are popular within the software community and a high proportion of software development companies use them in their development practice. The results however show that in the vast majority of cases, pattern usage is limited to design-based problems, involving a single type of pattern (i.e. design patterns). The results further show that process-based patterns are seldom used in the software development industry, which prompted the topic of the main investigation of this research to evaluate the effect and utility of process patterns.

A controlled experimental research method was designed and used to evaluate the utility and value of process patterns in the management of software development projects. In this '2x2 factorial design' experiment, the subjects were divided in two groups of experimental and control, where the experimental groups were given a set of process patterns to use in their software development projects. Overall, there were over 750 subjects involved in this experiment and a total of 260 software development projects (individual and group projects) were investigated. Measurements of a number of appropriate software attributes were taken during the life of the projects through a devised goal-based measurement process. A further number of attributes were measured after the projects were completed. Using metrics, a number of software attributes across the four major phases of the development lifecycle (i.e. Requirement Analysis, Design, Implementation, and Delivery) were measured and statistically analysed. In addition to these specific measurement data, official marks awarded to the projects by the tutors were also used in the analysis. The objective was to determine if the experimental groups produced software projects that were of higher quality, in terms of the measured software attributes, than the control groups.

The experiment results show that, in the case of thirteen measured attributes, the treated groups scored significantly higher than the control groups. The improvements are across all the four major development phases, with at least two attribute in each phase, showing significant improvement. The experiment, therefore, confirms that the application of process patterns in software development projects, improves the quality of the projects in terms of a number of specific attributes such as productivity and defect density. The results further show that the treated subjects in the group projects performed significantly better than those in the individual projects. This, therefore, confirms that while the application of process patterns significantly improves the quality of both group and individual projects, the improvement is more prominent in the case of team projects. Process patterns are thus shown to be more effective on team projects in improving the quality of software development projects.

---

## Table of Contents

<b>Chapter 1</b>	<b>Introduction and Outline</b>	<b>1</b>
1.1	<i>Introduction</i>	1
1.2	<i>The Pattern Concept</i>	1
1.3	<i>The Research Methods and Process</i>	2
1.3.1	Research Question and Hypothesis	3
1.3.2	Experimental Methodology	3
1.3.3	The Measurement Process	5
1.3.4	Results Presentation and Analysis	7
1.4	<i>Research Conclusions</i>	7
1.5	<i>Strengths of the Research</i>	8
1.6	<i>Thesis outline</i>	8
<b>Chapter 2</b>	<b>Software Engineering Patterns</b>	<b>10</b>
2.1	<i>Introduction</i>	10
2.2	<i>Software Engineering and Patterns</i>	10
2.2.1	How Patterns Entered Software Engineering	10
2.2.2	Software Pattern Definition	11
2.2.3	Pattern Elements and Types	11
2.2.4	What Patterns Are, and What They Are Not	12
2.2.5	Disregard for Originality	12
2.2.6	Characteristics of Patterns	13
2.2.7	Software Patterns and Pattern Principles	14
2.2.8	Software Pattern Usage in Industry	15
2.3	<i>Pattern Discussion</i>	15
2.3.1	Pattern Mining	15
2.3.2	Can Patterns be Harmful?	16
2.3.3	Do Software Patterns Work?	16
2.3.4	Should Patterns Be Formalised?	17
2.4	<i>Patterns in Software Design</i>	17
2.5	<i>Patterns in the Software Development Process</i>	19
2.6	<i>Instructions in Patterns</i>	24
2.6.1	Patterns in Town and Building Architecture	24
2.6.2	Patterns in Software Design	25
2.6.3	Patterns in Development Process	26
2.6.4	Software Process and Textual Instructions	28
2.6.5	Can Patterns Benefit from Task Analysis?	29
2.6.6	Hierarchical Task Analysis	29
2.6.7	Application of HTA in Patterns	31
2.6.8	Process Patterns Employed in the Experimentation	34
2.7	<i>Summary</i>	34
<b>Chapter 3</b>	<b>Pattern Usage Surveys</b>	<b>35</b>
3.1	<i>Introduction</i>	35
3.2	<i>Architectural Patterns Survey</i>	35
3.2.1	Motivation	35
3.2.2	Survey Details	35
3.2.3	Architectural Pattern Survey Results	36
3.3	<i>Survey of Software Organisations</i>	40
3.3.1	Motivations	40
3.3.2	Related Work	40

3.3.3	Samples and Sampling Method.....	41
3.3.4	Survey Instrument.....	41
3.3.5	Software Pattern Survey Results.....	42
3.4	Summary.....	49
<b>Chapter 4</b>	<b>Software Experimentation and Measurement.....</b>	<b>50</b>
4.1	Introduction.....	50
4.2	Measurement Theory and Definition.....	50
4.3	Purpose and Benefits of Software Measurement.....	52
4.4	Measurement Scales.....	53
4.5	Measurement Techniques.....	54
4.5.1	Direct and Indirect Measurement.....	54
4.6	Software Metrics.....	55
4.6.1	Process and Product Metrics.....	55
4.6.2	Composite/Hybrid Metrics.....	56
4.7	Measurement Validation.....	57
4.8	Software Quality Measurement.....	58
4.8.1	Factor Criteria Metric Models (FCM).....	58
4.8.2	Goal Question Metric Model.....	60
4.9	Measurement of Object-Oriented Software.....	61
4.10	Software Measurement Issues and Challenges.....	62
4.11	Experimentation in Software Engineering.....	65
4.11.1	Experimentation Framework.....	65
4.12	Software Experimentation Issues.....	66
4.12.1	Flaws in Experiment Design and Conduct.....	67
4.12.2	Subjects in the Experiments.....	67
4.12.3	Costs and Publishing Limitations.....	68
4.12.4	Human Factors.....	68
4.12.5	Experiment Quality.....	69
4.13	A Review of Pattern Related Experiments.....	69
4.14	Summary.....	72
<b>Chapter 5</b>	<b>Experimental Methodology.....</b>	<b>73</b>
5.1	Introduction.....	73
5.2	Experiment Definitions and Hypothesis.....	73
5.3	An Overview of the Experiment Design.....	74
5.4	An Overview of Issues Involved.....	76
5.4.1	Practical Difficulties.....	76
5.4.2	Ethical/Staff Concerns.....	77
5.5	Experiment Specification.....	77
5.5.1	Experimental Research Settings.....	77
5.5.2	Variables.....	77
5.5.3	The Treatment.....	79
5.5.4	Control.....	80
5.5.5	Internal Validity.....	82
5.5.6	External Validity.....	82
5.6	Experiment Design.....	83
5.6.1	Design models.....	83
5.6.2	Subject's Awareness of the Experiment.....	84

5.6.3	Subjects and Treatment Application .....	84
5.6.4	Subjects Selection Methods .....	85
5.6.5	Group and Individual Projects Assignments .....	86
5.7	<i>Experiment Conduct</i> .....	86
5.7.1	Application of Treatment .....	86
5.7.2	Data Types .....	87
5.7.3	Subjects' Views on Process Patterns.....	88
5.7.4	Experiment Outcome Scenarios .....	88
5.8	<i>Ethical Issues</i> .....	89
5.9	<i>Design Constraints</i> .....	90
5.10	<i>Summary</i> .....	91
<b>Chapter 6</b>	<b>Measurement Process .....</b>	<b>92</b>
6.1	<i>Introduction</i> .....	92
6.2	<i>Measurement Process Design</i> .....	92
6.2.1	GQM Tables.....	93
6.2.2	Metric Specifications.....	96
6.3	<i>Measurement Process Conduct</i> .....	98
6.3.1	Data Collection Procedure .....	98
6.3.2	Tools Used .....	99
6.4	<i>Summary</i> .....	99
<b>Chapter 7</b>	<b>Results .....</b>	<b>100</b>
7.1	<i>Introduction</i> .....	100
7.2	<i>Applied Statistical Methods</i> .....	100
7.2.1	Parametric Vs Non-parametric .....	100
7.2.2	Identification and Treatment of Outliers .....	100
7.2.3	Parametric Tests .....	101
7.3	<i>Teams Vs Individuals</i> .....	103
7.3.1	Further Analysis .....	103
7.4	<i>Sensitivity Analysis</i> .....	103
7.5	<i>Correlation/Regression Analysis</i> .....	104
7.5.1	Treatment Rate of Usage.....	105
7.6	<i>Conducted Measurement Results</i> .....	105
7.6.1	Requirements Analysis Phase .....	106
7.6.2	Design Phase .....	114
7.6.3	Implementation Phase .....	122
7.6.4	Delivery Phase .....	136
7.7	<i>Tutor Marks Results</i> .....	142
7.7.1	Product .....	142
7.7.2	Design and Analysis.....	144
7.7.3	Project Management.....	145
7.7.4	Evaluation .....	147
7.8	<i>Subjects' Views on Process Pattern</i> .....	149
7.9	<i>Summary</i> .....	150
<b>Chapter 8</b>	<b>Analysis .....</b>	<b>152</b>
8.1	<i>Introduction</i> .....	152
8.2	<i>Concise Results Representatian</i> .....	152
8.3	<i>An Analysis of the Results</i> .....	153



---

8.4	<i>Research Hypothesis</i> .....	159
8.5	<i>A Discussion of the Results</i> .....	159
8.5.1	Official Evaluation.....	160
8.5.2	Generalisations of the Results.....	160
8.6	<i>Summary</i> .....	161
<b>Chapter 9</b>	<b>Conclusion</b> .....	<b>162</b>
9.1	<i>Introduction</i> .....	162
9.2	<i>Summary of Main Concepts</i> .....	162
9.3	<i>Research Contributions</i> .....	164
9.3.1	Key Contribution.....	164
9.3.2	Additional Contributions.....	164
9.4	<i>Summary of Results</i> .....	165
9.5	<i>Limitations</i> .....	169
9.6	<i>Research's Impact</i> .....	170
9.7	<i>Future Work</i> .....	171
9.7.1	Software Experimentation.....	171
9.7.2	Patterns.....	171
	<b>Reference and Bibliography</b> .....	<b>173</b>
	<b>Appendix A. Experiment Details</b> .....	<b>192</b>
	<b>Appendix B. Patterns</b> .....	<b>199</b>
	<b>Appendix C. Metrics Specifications</b> .....	<b>210</b>
	<b>Appendix D. Results</b> .....	<b>227</b>
	<b>Appendix E. Survey Questionnaires</b> .....	<b>231</b>

## Table of Figures

Figure 2-1 Elements of a pattern (www.hillside.net) .....	12
Figure 2-2 Hierarchical structure of process patterns .....	21
Figure 2-3 Task process pattern for technical reviews .....	21
Figure 2-4 Hierarchical structure of process patterns .....	26
Figure 2-5 HTA for drawing a clock .....	31
Figure 2-6 Section of the goal hierarchy for an acid distillation plant operator's task [Annett 2004] ....	31
Figure 2-7 An example of task hierarchy for the Implementation phase .....	32
Figure 2-8 Example of a pattern sequence .....	33
Figure 2-9 An example of a pattern construct using HTA .....	33
Figure 3-1 Number of universities teaching architectural patterns .....	36
Figure 3-2 Architects' viewpoints in relation to pattern usage levels .....	37
Figure 3-3 Architects' viewpoints in relation to courses on patterns .....	37
Figure 3-4 Correlation between pattern usage and architect viewpoints .....	37
Figure 3-5 Companies using patterns .....	42
Figure 3-6 Pattern usage in relation to organisation size .....	42
Figure 3-7 Correlation between pattern usability and pattern usage .....	43
Figure 3-8 Correlation between pattern usefulness and pattern usage .....	43
Figure 3-9 Process patterns usage .....	45
Figure 3-10 Companies planning to use patterns .....	45
Figure 3-11 Companies developing patterns .....	46
Figure 3-12 Correlation between <i>reusability</i> and <i>pattern usage</i> .....	47
Figure 3-13 Correlation between <i>maintainability</i> and <i>pattern usage</i> .....	47
Figure 3-14 Correlation between pattern usage and testability-reliability quality attributes .....	48
Figure 4-1 A model of measurement [Oman and Pflieger 1997] .....	51
Figure 4-2 Measurement process and intelligence barrier [Kriz 1988] .....	52
Figure 4-3 Process pattern development through process improvement .....	53
Figure 4-4 Factor-Criteria-Metrics general model .....	59
Figure 4-5 An example of FCM model for maintainability .....	59
Figure 4-6 Factor/Criteria/Metrics model (McCall/Boehm model) .....	59
Figure 4-7 The Goal Question Metric Model .....	60
Figure 4-8 V-GQM Model .....	60
Figure 4-9 SATC Model for Software Metrics Programme .....	61
Figure 5-1 Experiment Design .....	75
Figure 5-2 Capture and analysis of data to test the research hypothesis .....	76
Figure 5-3 Many-to-many relationship between process patterns and metrics .....	80
Figure 5-4 Random subject selection .....	81
Figure 5-5 Matching by precision control technique .....	81
Figure 5-6 Experiment design .....	83
Figure 5-7 Module CMT3991 (group projects) seminar structure .....	86
Figure 5-8 An outcome Scenario .....	88
Figure 5-9 An outcome Scenario .....	88
Figure 5-10 An outcome scenario .....	89
Figure 5-11 An outcome scenario .....	89
Figure 7-1 Regression line .....	104
Figure 7-2 Regression line scatter plot .....	104
Figure 7-3 Rate logins to the treatment (i.e. process patterns) website .....	105
Figure 7-4 Boxplot for percentage of traceable requirements .....	106
Figure 7-5 Correlation between the no. of logins and traceable requirements .....	107
Figure 7-6 Correlation between the no. of logins and traceable requirements for individual projects .....	107
Figure 7-7 Boxplot for percentage of requirements specification reviewed .....	109
Figure 7-8 Boxplot for the percentage of defects fixed in RA .....	111
Figure 7-9 Boxplot for percentage of RA time spent in testing .....	113
Figure 7-10 Boxplot for the percentage of design document reviewed .....	115
Figure 7-11 Boxplot for No. of Methods per Class .....	117
Figure 7-12 Boxplot for the percentage of defects fixed in design phase .....	119
Figure 7-13 Boxplot for percentage of Design phase time spent in testing .....	121
Figure 7-14 Boxplot for Comment Density (Com/100LOC) .....	123
Figure 7-15 Boxplot for percentage of source code reviewed .....	125

---

Figure 7-16 Boxplot for defect density.....	127
Figure 7-17 Boxplot for productivity in the Implementation phase .....	129
Figure 7-18 Boxplot for overall productivity.....	131
Figure 7-19 Boxplot for the percentage of defects fixed in the Implementation phase.....	133
Figure 7-20 Boxplot for percentage of implementation time spent in testing.....	135
Figure 7-21 Boxplot for test case density .....	137
Figure 7-22 Boxplot for percentage of defects fixed in the Delivery phase.....	139
Figure 7-23 Boxplot for percentage of Delivery phase time spent in testing.....	141
Figure 7-24 Boxplot for the product attribute .....	143
Figure 7-25 Boxplot for the Design and Analysis marked attribute .....	144
Figure 7-26 Boxplot for project management marked attribute.....	146
Figure 7-27 Boxplot for the evaluation attribute .....	148
Figure 7-28 process pattern usefulness.....	150
Figure 7-29 Process patterns usability .....	150
Figure 9-1 Correlation between reusability and pattern usage .....	166
Figure 9-2 Correlation between maintainability and pattern usage.....	166
Figure App_A 1 Snapshots of online measurement form .....	192
Figure App_B 1 Login form.....	201
Figure App_B 2 Snapshot of a process patterns hosted online for the experiment.....	201
Figure App_B 3 Snapshot of a process patterns hosted online for the experiment.....	202

## Table of Tables

Table 1-1 The analysed metrics and tutor marks.....	7
Table 2-1 Pattern sequence to add support for service interfaces [Siddle 2007] .....	14
Table 2-2 Results summary [Beck et al. 1996].....	15
Table 2-3 GoF's design pattern elements [Gamma et al. 1995] .....	18
Table 2-4 Prototype process pattern .....	20
Table 2-5 Elements of process pattern [Ambler 1998].....	21
Table 2-6 Pattern elements in [D'souza and Wills 1999] patterns .....	22
Table 2-7 Pattern elements [Storrie 2000] patterns .....	23
Table 2-8 Pattern elements in Cary and Carlson [2002].....	23
Table 2-9 Window Place Pattern.....	25
Table 2-10 Outdoor room Pattern .....	25
Table 2-11 An example of a pattern sequence for building a porch .....	25
Table 2-12 Model-View-Controller.....	26
Table 2-13 Decorator Pattern .....	26
Table 2-14 Requirement Analysis Pattern.....	27
Table 2-15 Big Ball of Mud process pattern .....	27
Table 2-16 A language for object development from scratch.....	27
Table 2-17 Example of a process function (program) Osterweil [1987].....	28
Table 2-18 HTA notations.....	30
Table 2-19 Process Pattern language.....	32
Table 3-1 Correlation between pattern usage and viewpoints.....	37
Table 3-2 An example of the survey questions .....	41
Table 3-3 Pattern usability results .....	42
Table 3-4 Pattern Usefulness Results .....	43
Table 3-5 Reasons for not using patterns .....	45
Table 3-6 Participants' viewpoints on the effect of patterns on quality attributes .....	47
Table 3-7 Correlation analysis for testability, reliability, and pattern usage.....	48
Table 3-8 Patterns effect on communication.....	48
Table 4-1 Measurement scale types .....	54
Table 4-2 Examples of indirect measures .....	55
Table 4-3 Examples of Internal and external attributes for products .....	55
Table 4-4 CK metrics.....	62
Table 4-5 Negative aspects of software measurement [Hall et al. 2001].....	63
Table 4-6 Elements of the definition phase.....	66
Table 4-7 Elements of the planning phase.....	66
Table 5-1 Experiment arrangements for the group projects.....	74
Table 5-2 Experiment arrangements for the individual projects.....	74
Table 5-3 Experiment design.....	75
Table 5-4 The independent variables.....	78
Table 5-5 The 2 × 2 experiment design (independent variables).....	84
Table 5-6 Relationships between the development phases and the marked attributes .....	87
Table 6-1 Goal Elements of the GQM model .....	93
Table 6-2 GQM for artefacts in the Requirement Analysis (RA) phase .....	94
Table 6-3 GQM for test and review in the RA phase .....	94
Table 6-4 GQM for effort in the RA phase.....	94
Table 6-5 GQM for artefacts in the Design phase.....	94
Table 6-6 GQM for test and review in the Design phase .....	95
Table 6-7 GQM for effort in the Design phase .....	95
Table 6-8 GQM for artefacts in the Implementation phase .....	95
Table 6-9 GQM for test/review in the Implementation phase.....	95
Table 6-10 GQM for effort in the Implementation phase.....	96
Table 6-11 GQM for artefacts in the Delivery phase .....	96
Table 6-12 GQM for test/reviews in the Delivery phase.....	96
Table 6-13 GQM for effort in the Delivery phase .....	96
Table 6-14 Percentage of traceable requirements metric (Metric 1).....	97
Table 6-15 Number of traceable requirements measure (Measure 1).....	97
Table 6-16 Number of requirements measure (measure 2).....	98
Table 7-1 Relationships between experiment groups and semesters .....	102

Table 7-2 Statistics for percentage of traceable requirements.....	106
Table 7-3 Statistics for the percentage of the requirements specification reviewed.....	108
Table 7-4 Results tables layout.....	109
Table 7-5 Statistical analysis for the 'percentage of reviewed requirements specification' metric.....	110
Table 7-6 Statistics for the percentage of defects fixed in RA phase.....	111
Table 7-7 Statistical significance analysis for the 'percentage of defects fixed' metric.....	112
Table 7-8 Statistics for the percentage of RA phase time spent in testing.....	112
Table 7-9 Results of significance analysis.....	114
Table 7-10 Statistics for the percentage of design document reviewed.....	115
Table 7-11 Statistical significance analysis.....	116
Table 7-12 Statistics for the no. of methods per class.....	117
Table 7-13 Statistical significance analysis for the 'No. of methods per class' metric.....	118
Table 7-14 Statistics for the percentage of defects fixed in the design phase.....	119
Table 7-15 Statistical significance analysis for the 'percentage of defects fixed' metric.....	120
Table 7-16 Statistics for the percentage of the Design phase time spent in testing.....	121
Table 7-17 Statistical significance analysis.....	122
Table 7-18 Statistics for the Comment Density.....	123
Table 7-19 Statistical significance analysis for the 'Comment density' metric.....	124
Table 7-20 Statistics for the percentage of source code reviewed.....	125
Table 7-21 Statistical significance analysis for the 'percentage of source code reviewed' metric.....	126
Table 7-22 Statistics for the defect density in the source code.....	127
Table 7-23 Statistical significance analysis for the 'defect density' metric.....	128
Table 7-24 Statistics for productivity in the Implementation phase.....	129
Table 7-25 Statistical significance analysis for the 'Implementation productivity' metric.....	130
Table 7-26 Statistics for the overall productivity.....	131
Table 7-27 Statistical significance analysis for the 'overall productivity' metric.....	132
Table 7-28 Statistics for the percentage of defects fixed in the Implementation phase.....	133
Table 7-29 Statistical significance analysis for the 'percentage of defects fixed' metric.....	134
Table 7-30 Statistics for the percentage of Implementation phase time spent in testing.....	135
Table 7-31 Results of significance analysis.....	136
Table 7-32 Statistics for test case density in the Delivery phase.....	137
Table 7-33 Statistical significance analysis for the 'test case density' metric.....	138
Table 7-34 Statistics for the percentage of defects fixed in the Delivery phase.....	139
Table 7-35 Statistical significance analysis for the 'percentage of defects fixed' metric.....	140
Table 7-36 Statistics for the percentage Delivery phase time spent in testing.....	140
Table 7-37 Statistical analysis for the 'percentage of phase time spent in testing' metric.....	141
Table 7-38 Relationships between the development phases marked attributes.....	142
Table 7-39 Statistics for the product attribute.....	142
Table 7-40 Statistical significance analysis for the 'product' attribute.....	143
Table 7-41 Statistics for the design and analysis marked attribute.....	144
Table 7-42 Statistical analysis for the 'design and analysis' marked attribute.....	145
Table 7-43 Statistics for the <i>project management</i> marked attribute.....	146
Table 7-44 Statistical analysis for the 'Project Management' attribute.....	147
Table 7-45 Statistics for the <i>evaluation</i> attribute.....	148
Table 7-46 Statistical analysis for the <i>evaluation</i> attribute.....	149
Table 8-1 A concise representation of metrics/marks results.....	153
Table 8-2 Metrics that showed positive effect of process patterns and their effect size.....	154
Table 8-3 Metrics that showed no significant effect of process patterns.....	154
Table 8-4 Metrics that showed process patterns had a more significant effect.....	158
Table 8-5 Tutor mark attribute, which showed positive effect of process patterns.....	159
Table 9-1 Summary of the results.....	167
Table 9-2 Improved attributes and the effect size.....	167
Table App_A 1 Official marking criteria for group and individual projects.....	196
Table App_A 2 Grading arrangements for <i>product</i> criteria.....	196
Table App_A 3 Grading arrangements for <i>evaluation</i> criteria.....	197
Table App_A 4 Grading arrangements for <i>Design and Analysis</i> criteria.....	197
Table App_A 5 Grading arrangements for <i>Project management</i> criteria.....	197
Table APP_C 1 A description of the elements of the metric specification table.....	210

---

Table App_D 1 Significance analysis results for metrics .....	228
Table App_D 2 Significance analysis results for tutor marks .....	228

### **Table of Equations**

Equation 4-1 Weighted Method per Class.....	62
Equation 7-1 z-score .....	101
Equation 7-2 Independent samples t-test.....	102
Equation 7-3 Correlation coefficient .....	104

### **Table of Metrics (Indirect Metrics)**

Metric 1 Percentage of Traceable Requirements.....	211
Metric 2 Percentage of Defects Fixed .....	212
Metric 3 Percentage of Requirement Specification Document Reviewed.....	213
Metric 4 Percentage of Phase Time Spent on Testing.....	214
Metric 5 Percentage of Design Document Reviewed.....	215
Metric 6 Methods per Class Ratio .....	216
Metric 7 Productivity .....	217
Metric 8 Percentage of Source Code Reviewed .....	218
Metric 9 Defect Density .....	219
Metric 10 Comment density.....	220
Metric 11 Test Case per Requirement Ratio .....	221

### **Table of Measures (Direct Metrics)**

Measure 1 Number of Traceable Requirements .....	222
Measure 2 Number of Requirements .....	222
Measure 3 Number of Detected Defects.....	223
Measure 4 Number of Defects Fixed.....	223
Measure 5 Time Spent in a Development Phase.....	223
Measure 6 Total Time Spent on Development Project .....	224
Measure 7 Time Spent in Testing in a Development Phase .....	224
Measure 8 Size of Source Code (LOC).....	224
Measure 9 Number of Classes.....	225
Measure 10 Number of Methods .....	225
Measure 11 Number of Lines of Comment .....	225
Measure 12 Number of Defined Test Cases .....	226

---

## Chapter 1 Introduction and Outline

---

### 1.1 Introduction

With continuing advances in computing hardware, software can now be produced to simulate and automate many complex human activities, thus making software development more complex and challenging. Therefore, the capture and preservation of experience and 'best practice' in software development, in terms of both product and process, is essential for the purpose of reuse. The *pattern* concept proposed by Alexander [1977, 1979] provides a way of preserving such experience. This research aims to investigate whether patterns are effective in enhancing the quality of software development projects.

This chapter provides an introduction to the thesis and presents the research process through a streamlined discussion. Specifically, this chapter presents the following main topics:

- Background knowledge on patterns
- Research methods and process
- Research question and hypothesis
- Research Conclusions
- Thesis structure

These topics will be discussed in the following sections.

### 1.2 The Pattern Concept

It was the work of a team of researchers in the field of town and building architecture in the 1960's and 1970's, and the philosophy of aesthetic and beauty in architecture that created the concept of *pattern*. The researchers realised that there were repeating elements in great architectural structures, such as cathedrals and monasteries, which made them pleasing to the eye and created feelings of joy and satisfaction to their observers. They called these elements *patterns* and introduced a way of capturing and documenting them [Alexander 1977, 1979].

As software designs and methodologies were becoming more complex and versatile during the 1980's, it was realised that it was necessary to preserve and present proven software designs in a systematic and methodological manner. This made some researchers look into other disciplines for solutions. The work of Alexander [1977, 1979] on the pattern concept inspired some researchers to adopt the concept in software engineering. Alexander was already known in the software community with his work on the 'synthesis of the form' [Alexander 1970], which played a key part in inspiring the Object Oriented paradigm in software engineering. Researchers working on software development designs in the early 1990s found that the pattern concept could help simplify some of the complexities involved in designing and developing software applications. Based on Alexander's pattern theory [Alexander 1977, 1979], they produced a number of patterns that described solutions to a number of design problems. These patterns, named *design patterns*, were well received in the computer science community and produced some excitement. Since 1994, many pattern conferences (under the name PLoP – Pattern Language of programming) have been established all over the world and numerous journal papers and books on patterns have since been produced.

In its simplest form, a pattern describes the solution to a problem in a context [Coplien 1995]. Alexander [1977] states the following to describe patterns: "Each pattern is a three-part rule, which expresses a relation between a certain context, a problem, and a solution". Patterns have the following fundamental characteristics [Winn and Calder 2002]:

- Present proven solutions and therefore cannot represent new concepts, designs, or solutions
- Concern both product and process
- Have a human element
- Relate to other patterns to create a pattern language for a specific domain

Patterns have been used in many areas within software engineering including development, organisation, software process, software modelling, etc [PLoP conferences 1994-2007]. Studies (e.g. [Manolescu et al. 2007]) have shown that patterns in the field of software design, referred to as *design patterns*, are the most widely used patterns in the software industry. These patterns are concerned with coding and architecture design of programs. There are copious works in terms of books and research papers on design patterns (e.g. PLoP 1994 – 2007 conferences, and journals such as IEEE software). Two of the most respected works on patterns in software design and architecture are by Gamma et al. [1995] and Buschmann et al. [1996]. These and other works are reviewed in Chapter 2. There have been many claims in the literature that design patterns improve software quality [Gamma et al. 1995] [Buschmann et al. 1996]. Over the years, there has been some empirical research to investigate the validity of such claims [Prechelt et al. 2002, 2001]. These investigations and their results are discussed in the literature review chapters (Chapter 2, Chapter 4).

In addition to the use of the pattern concept in the technical design of software (i.e. product patterns), patterns have also been applied in development process. These patterns, referred to as *process patterns*, have interested researchers right from the outset of the pattern movement in the early 1990's. While product patterns (e.g. design patterns) describe the solution, process patterns describe the process that leads to the desired results. They document proven process activities in a structured and consistent manner, in accordance with the pattern concept and format. Coplien [1995] and Ambler [1998] define process patterns as “Patterns of activity within an organisation and its projects”, and “Patterns which describe a proven solution, successful approach and/or series of actions, for developing software” respectively. Pressman [2005] states that process patterns provide a consistent method for describing important characteristics of the software process. There are many published works [Coplien 1995] [Ambler 1998, 1999] on process pattern, some of which will be discussed and reviewed in Chapter 2. As well as the PLoP conferences, there have also been workshops on process patterns [SDPP 2002], where papers have been presented and subsequently published in their proceedings.

Process patterns, in particular, are the subject of investigation in this research programme. Pattern authors have written and produced numerous process patterns published in books, journals and conference proceedings (key works are reviewed in Chapter 2). Many authors have claimed that the use of process patterns in software development improved software quality [Coplien 1995] [Ambler 1998]. Pressman [2005] states that process patterns provide an effective mechanism for describing any software process and, that by combining process patterns, a software team can construct a process which best meets the needs of a project. However, there does not appear to be any empirical studies substantiating claims of the beneficial utilities of process patterns. It has been the main objective of this research programme to investigate such claims and determine if the application of process patterns in software development projects does indeed improve the quality of the projects.

### 1.3 The Research Methods and Process

Having briefly discussed the motivations for the research and, the underlying pattern concept, the research method and process is discussed in this section. A number of research methods were studied and considered for employment in this study. These included *experimental*, *survey*, *existing data*, *action research*, *meta-analysis*, and *case study*, research methods.

The key features of the experimental approach are manipulation and control. In order to test hypotheses, changes are introduced into the environment, in the form of treatment, and the effects of the changes on the target are observed or measured. The strength of this research method is that it makes it possible to exercise greater control over the conditions of observation than in any other research method and, therefore, experimental studies have long been regarded as the optimal way to test causal hypotheses [Singleton and Straits 1999]. Based on these characteristics and its suitability to test the research's main hypothesis, the experimental research method was chosen as the main research method in this study. A further reason for the selection of this research method was its suitability to provide a validation strategy for patterns.

Survey research provides the mechanism for acquiring information about one or more groups of people or organisations. The ultimate goal is to learn about a large population by surveying a sample of that population [Leedy and Ormrod 2005]. Surveys are typically administered in face-to-face, telephone interviews, paper-and-pencil, or electronic questionnaires. The validity and reliability of the survey instrument are key considerations in survey research. Validity refers to the extent to which an empirical measure adequately reflects the real meaning of the concept under consideration, and reliability is a matter of whether a particular technique, applied repeatedly to the same object, would yield the same result each time [Babbie 2001]. The survey research method was used in this study to assess the popularity and usage levels of patterns within software and architecture communities.



The use of existing rather than newly generated data provides a further research method. In this research method, the researcher saves time by using the existing data rather than collecting new ones. Scientific research should not be equated with the collection and analysis of original data [Babbie 2001]. In many situations, scientific research can be designed and conducted through analysis of data already collected and compiled. As existing data, the official marks awarded to the student projects were used in testing the research hypothesis.

### 1.3.1 Research Question and Hypothesis

A literature review process was undertaken in an attempt to narrow down the field of research on software patterns and develop a research question. The literature review showed that there was a dearth of empirical research that presented evidence of practical utility and value of software patterns. This review also indicated that the pattern concept received a mixed reception within the architecture community, in which the concept was originally conceived.

In addition to the literature review, two preliminary surveys were conducted to help understand the current state of affairs regarding the pattern concept, both within the software engineering and architecture communities. The first survey, gauged the views of a number of UK software development companies, 67 of which responded [Estabraghy and Dalcher 2007b]. The second survey gauged the views of architects in UK universities on the use and value of architectural patterns. These preliminary surveys were aimed, not only to provide evidence of pattern usage, but also to help in constructing and setting a research question (see Chapter 3). The result of these surveys, in addition to the results of the literature reviews, pointed to a lack of published empirical research on the practical application of process patterns.

Based on the results of the surveys and the literature reviews, the main goal of the research was set to evaluate the utility and value of process patterns in the management of software development projects. The aim was to determine whether process patterns would improve the quality of a software projects that use them. The research question posed, therefore, was:

*How does the application of process patterns in the management of a software development project affect the quality of the project?*

Based on the research question, the null and alternative hypothesis to be tested was:

- $H_0$       Application of process patterns in the management of a software development project *will not* improve the quality of the project
- $H_1$       Application of process patterns in the management of a software development project *will* improve the quality of the project

In investigating the research question and the corresponding hypothesis, the effect of process patterns on each of the four major development phases (Requirement Analysis, Design, Implementation, and Delivery) of a software development project was investigated through measurement and evaluation of a number of software attributes in an experimental research method. The term software project is used throughout this thesis to refer to the mentioned four phases and the process and product involved in each. It does not include project activities such as planning, scheduling, or estimation.

A controlled experimental research method, based on its suitability as discussed in previous section (Section 1.3), was designed and implemented to test the hypothesis. The process is described in the following section.

### 1.3.2 Experimental Methodology

Although patterns have been an important and influential component of software engineering in the last decade [Buschmann 2007b], empirical studies on the effects and utility of patterns are rather rare. While there has been some experimental research to evaluate the effect and value of design patterns [Prechelt 2001, 2002], there appeared to be no credible published empirical studies to investigate the utility and value of process patterns. This study addressed this issue by carrying out an empirical study to evaluate the effect and value of process patterns by investigating the hypothesis that the deployment of process patterns in software development results in better software development projects.

A literature survey of software experimentation indicated that, ideally, the experimental research method would be best conducted in an industrial setting. However, due to the lack of finding suitable industrial organisations willing to participate in this experiment, an option of conducting the experiment using students working on practical projects in an academic institution, such as Middlesex University, was considered. The literature survey also indicated that majority of experimental researches in software engineering had used students as subjects and such experiments are recognised by the scientific community as valid. Sjoberg et al. [2005] in their detailed analysis of empirical software engineering found that 75% of the subjects in empirical investigations were students.

Middlesex University was therefore selected as the setting for the experiment. Two course modules in software project management for final-year undergraduates, that included practical software development projects, were selected. Students taking the modules over two semesters were chosen as the subjects of the experiment. However, using students and live course modules as subjects and objects of an experiment involved tackling many inherent issues, particularly as this was the first time such an experiment was to take place at the University and there was no precedence. An experiment design had to be designed which gained the approval of the Ethics Committee as well as the course officials. Much negotiation, discussion, and design revision had to take place before the experiment design was approved by the Ethics Committee and permission was granted to carry out the experiment.

The devised controlled experiment involved two types of software projects (i.e. individual and group projects) for the final year undergraduate students across two successive semesters. For each project type, students were divided into two groups of experimental and control groups where the experimental groups received a number of process patterns (98 in total), covering a complete development lifecycle (from Requirement Analysis to Delivery). The process patterns (i.e. treatment condition) were hosted on a specifically designed website to be used by the experimental groups in doing their software development project. The process patterns were selected from the literature, based on their suitability and appropriateness for the type and scope of the projects being investigated in this experiment. The selected process patterns were further edited to make them concise, relevant, and applicable to the type and scope of the projects used in the experiment. The frequency of access to the process patterns for each subject was recorded and continuously monitored to ensure that materials were accessed by experimental groups. The control groups were not given access to these process patterns to use in their projects. Official project assignments were devised and prepared in cooperation with the module leader, coordinators, and tutors, so that the experimental groups were required to use the given process patterns in their projects. Throughout the two semesters, the researcher was actively involved in attending the relevant modules' lectures and seminars, to answer any questions and queries on process patterns, to ensure that students (experiment subjects) understood how to use the process patterns and that they were actively using them.

In evaluating the process patterns, two strategies were considered: 1) Evaluation of a small number of individual and specific process patterns, and 2) Evaluation of a system of process patterns covering a complete development lifecycle. While option (1) initially appeared to be preferred for its specificity and simplicity, it suffered from the following two disadvantages:

- Process patterns are generally linked and related to each other and it is often impractical to isolate individual process patterns and evaluate their effect on specific software quality attributes.
- It limits the scope of the study to specific process patterns, rather than a complete system of patterns. Any results would therefore apply to those specific patterns rather than process patterns in general.

It was therefore decided to select and implement option (2), to study the effect of a complete system of process patterns covering the complete development lifecycle (i.e. Requirement Analysis, Design, Implementation, and Delivery). The objective of the experiment was, therefore, not to determine whether the employment of any particular process pattern had an effect on the quality of a software project, but to gauge the collective influence of a whole system of process patterns. In this design, one or more software attributes could be affected by one or more process patterns. There is therefore a many-to-many relationship between the process patterns and the software attributes that they could affect. That is, more than one process pattern can affect the value of a single attribute, and more than one attribute can be affected by a single process pattern (see Section 5.5.3).

The process patterns used as treatment in the experiment cover a complete development lifecycle and therefore their possible effect is measured on the major development phases. In testing the experiment's hypothesis, the objective was to determine whether the development projects carried out by the treated groups were of better quality than those carried out by control groups. The evaluation of the development projects were performed

through a number of metrics based on collected measurements about the projects, as well as official marks given to the projects by the project tutors and supervisors.

The key objective in these metrics was to determine any difference between the treated and control groups by keeping the rules constant for both groups, with the treatment (i.e. process patterns) being the only differentiating variable. The metric values were therefore used in this experiment in the context of comparing treated and control groups. The metrics were devised to be applicable in the set environment for the comparison purposes of the experiment, and did not require being necessarily generic. For example, defects are normally measured in terms of 'defect per 1000 line of code' at industry level applications. However, due to the small size of the developed software in the projects under investigation, the defects were measured in 'defect per 100 lines of code'. Furthermore, while a flawless and well designed and conducted measurement process was desirable and advantageous, many possible weaknesses and flaws in the employed measurement processes would be of no serious harm to the objectives of the experiment, since they would be constant and equivalent for both treated and control groups.

There were two sets of distinct and independent measurement data used in the experiment:

- 1. Official marks**

Projects were marked by tutors on a number of attributes (12 in all). While most of these attributes were concerned with the actual project report (i.e. abstract, introduction, conclusion...), there were some (i.e. Design and analysis, product, evaluation, project management) which were directly related to the development efforts in which this study was interested. These marks were made available to the researcher and were used for the purpose of this study.

- 2. Collected measurements**

A measurement process was devised and conducted to capture specific measurements on a wide range of attributes, in accordance with the experiment's goals. Such measurements were taken by the subjects, during the lifetime of their projects and, were submitted through online forms for analysis. Further measurements of a number of attributes were taken by the researcher by evaluating the completed project reports.

The measurement process aimed to develop a tailor-made measurement plan, through which a number of appropriate metrics would be devised, to evaluate the projects. This is briefly discussed in the next section.

### 1.3.3 The Measurement Process

Software measurement is not yet an exact science. While the software community agrees that measurement should be an important activity in software development and engineering, there are disagreements on what and how to measure software projects. Furthermore, while one can intuitively recognise quality attributes, there is a lack of a universally accepted definition of software quality that can be accurately measured [Kitchenham and Pfleeger 1996] [Ebert and Dumke 2007].

The literature review carried out indicated that software quality has been largely measured in terms of the end product and process, rather than in terms of the quality of the individual development phases in a complete development lifecycle (see for example [Briand et al. 2001]). A distinguishing aspect of this experiment is that, the quality of the development project in all its major individual phases (i.e. Requirement Analysis, Design, Implementation, and Delivery) were investigated. Here are the characteristics of the devised measurement process in this research:

- The measurement process was goal-oriented and was based on the Goal/Question/Metric (GQM) model
- Each major development phase was individually evaluated
- Software attributes related to each development phase were accessed.

There are numerous software metrics in the literature, developed for different purposes. While some of these metrics can be reused in other projects, the uniqueness of projects normally necessitates a study of the measurement requirements specific to the projects under study [McGarry 2001]. A goal-oriented measurement process, using the Goal/Question/Metrics paradigm [Basili and Rombach 1988], was devised for generating the required metrics and collecting/recording the required measurements for the experiment. The purpose of the devised metrics was to measure the quality of the software projects under investigation, through the measurement of a number of attributes, to evaluate the effect and utility of process patterns. For each

development phase, the devised metrics were used to measure the following categories of the process and product attributes:

- **Artefacts:** Artefacts (such as code and documents) produced during each development phase
- **Tests/reviews:** The testing/reviewing quality of each development phase
- **Effort:** The proportion of time allocated to each phase

Two methods of collecting and recording data were used in the measurement process:

1. **Measurements taken by the experiment subjects:** These measurements were taken by the subjects, during the life of their projects, and submitted through a specific online measurement form.
2. **Measurements taken by the researcher:** This was done after the completion of the projects and their assessments by the module tutors. The researcher studied each project report and recorded a number of measures (e.g. number of traceable requirements) for each project.

In addition to the two measurement types stated above, the tutor marks provided a further set of measurement data that were used and analysed. The comprehensive set of measurements aimed to provide wide-ranging data to enable the evaluation of various aspects of the development projects under investigation (in terms of both process and product) for any differences between the control and treated groups. Due to the limitation on the scope of the research, only a proportion of the measurement data were used for presentation and analysis in this thesis report.

A number of options for evaluating and analysing the measurements in terms using appropriate metric types were considered. These are as follows:

- 1) Using the direct and indirect non-composite and non-hybrid metrics (e.g. 'defect density')
- 2) Devising a new strategy of combining related metrics to create composite/hybrid metrics to measure the quality attributes.

While option (2) has the advantage of generating high level metrics that produces overall evaluation of multi-faceted attributes, it is a complex method of normalising and combining metrics of different types compositely and may have the disadvantage of producing less sensitive results (see Section 4.6.2). Option (1) was therefore chosen for its advantages of greater simplicity and wider use.

Measurement data collected through the experiment were used to draw up a number of metrics to measure a number of software attributes of interest. There were also a number of attributes, marked by the tutors, which were used in the experiment. The Table 1-1 list the metrics and tutor marks used in the experiment. The metrics were aimed to measure attributes across the four major phases of the development lifecycle. There are, in all, 18 metrics involved. The result and analysis of the metrics and tutor marks will be presented in Chapter 7 and Chapter 8.

Metrics	Req. Analysis	Percentage of traceable requirements
		Percentage of reviewed requirements specification
		Percentage of defects fixed
		Percentage of phase time spent in testing
	Design	Number of methods per class (Methods per Class Ratio)
		Percentage of design document reviewed
		Percentage of defects fixed
		Percentage of phase time spent in testing
	Implementation	Comment density
		Percentage of code reviewed
		Percentage of defects fixed
		Productivity (Implementation Phase)
		Productivity (Overall)
Defect density		
Delivery	Percentage of phase time spent in testing	
	Test case density (Test case per Requirement)	
	Percentage of defects fixed	
Marks	Percentage of phase time spent in testing	
	Design and analysis	
	Product	
	Evaluation (tests)	
		Project management

Table 1-1 The analysed metrics and tutor marks

Having discussed the measurement process through which metrics were devised and measurements were collected, the presentation and analysis of the metrics results are briefly explained in the next section.

### 1.3.4 Results Presentation and Analysis

The values of the metrics, developed through the measurement process for each major development phase, were presented and analysed for statistical significance using SPSS statistical analysis package. It was also analysed as to whether there were any difference between the group projects and individual project in terms of the evaluated metrics. Both, 2x2 Factorial ANOVA and independent samples t-test, statistical methods were applied to analyse the metrics values and judge their statistical significance. Further analysis in terms of correlations between metric values and logins to the online process patterns, as well as sensitivity analysis of the metrics, were also carried out. Once the experimental research method was implemented and data was collected and analysed, they would be presented in a thesis report.

## 1.4 Research Conclusions

The survey of the software industry indicated that while design patterns were being used in industry regularly, little was known about process patterns and, its usage in industry, was shown by the survey to be relatively low [Estabraghy and Dalcher 2007b]. Many participants stated that software patterns improved software attributes such as reusability, reliability, and maintainability. The survey of architects indicated that architectural patterns were seen by many architects as an old fashioned and anti-creativity concept. It was discussed that the architectural pattern issues, such as anti-creativity, which were the major causes of their unpopularity, need not necessarily be applicable to, or damage the utility of, software patterns.

Analysis of the measurement data confirmed the main hypothesis, that the application of process patterns in the management of software development projects, improved the quality of the projects. It showed that thirteen measured software attributes were improved as a result of using process patterns. The analysis of the conducted measurement showed that, for the majority of the evaluated attributes, there was a statistically significant difference between projects that used process patterns and those that did not [Estabraghy and Dalcher 2007a]. The difference between the treated and control groups indicated that the treated groups performed better in all the four development phases investigated. The analysis of the marks awarded to the projects, by the project tutors and supervisors, showed that there was a statistically significant difference between the treated and control

groups for one (i.e. product) of the four development attributes marked (i.e. Design and Analysis, Product, Evaluation, Project management).

The results also showed that, for many metrics as well as the product attribute (as marked by tutors), the effect of the treatment condition was higher on group projects than on individual projects. This indicates that the process patterns have a more prominent effect on team projects than on individual projects. The results also showed that the majority of the subjects that used patterns (treated groups) found process patterns useful and easy to use.

## 1.5 Strengths of the Research

In this research, two preliminary surveys were carried out on software patterns where there is scant previous research. Only one recent publication [Manolescu et al. 2007] reports on a similar research survey. The surveys investigate the architectural and software patterns, in terms of their popularity, in their respective communities and discuss the reasons software patterns have been much more utilised and successful than architectural patterns. There does not appear to be any published literature that has previously explored this topic.

There are a number of attributes that make the designed and implemented-controlled experiment of high quality in comparison to other software engineering experiments. These include:

*Controlled Experiment:* Controlled experiments are often expensive and difficult to conduct and therefore only a small proportion (1.9%) of software experimentations are controlled experiments [Sjoberg et al. 2005]. Hypothesis testing is also rare at around 1% of the software experiments [Tichy et al. 1995].

*Sensitivity Analysis:* In the literature review carried out in this research, the proportion of software experiments that included sensitivity analysis was found to be extremely low. Except in a very few specific cases, sensitivity analysis was not found to be routinely applied in the published software engineering experiments.

*Number of experiment subjects:* The average number of subjects used in software experiments is 49 [Sjoberg et al. 2005]. This experiment involved a total of 752 subjects.

*The experiment duration:* The experiment was of a relatively long duration. It had two phases, which spanned two semesters (a total of six months).

*Real situation experiment:* This experiment was done based on real final-year undergraduate student projects (i.e. not for the purpose of the experiment only).

*Coverage and evaluation of a complete development lifecycle:* The experiment evaluated attributes from a complete development lifecycle.

*Detailed Statistical Analysis:* Referential, as well as descriptive statistical analysis was carried out to statistically present and analyse the results.

In the next section, the outline of the thesis will be discussed.

## 1.6 Thesis outline

The thesis contains nine chapters, which are briefly described in this section.

### Chapter 1 - Introduction

This chapter offers an introduction to the thesis. As well as providing the general layout of the thesis, it briefly presents the background knowledge and describes the study's research methods and process. The chapter also presents the research question and the research hypothesis to be tested in this study.

### Chapter 2 - Software Patterns

Software patterns are the key element of this research project. This chapter therefore discusses the concept of pattern and the issues involved. The chapter presents a literature review of the most relevant research. The chapter also discusses whether task analysis could be utilised in developing and sequencing patterns.

---

**Chapter 3 – Pattern Usage Surveys**

Two preliminary surveys were designed and conducted in an attempt to understand the pattern issues and to help derive the research question for this study. The chapter discusses why, while architectural patterns suffer from a number of criticisms and deficiencies, software patterns are not affected. The chapter discusses the popularity of both architectural and software patterns as indicated by the conducted surveys.

**Chapter 4 - Software Measurement and Experimentation**

Both software measurement and experimentation are important components of this research. This chapter therefore presents and discusses the backgrounds to these two topics and reviews the related literature. The chapter further discusses the current issues and difficulties in both software experimentation and software measurement.

**Chapter 5 - Experimental Research Method**

The experimental methodology is the main research method of this study. This research method is used to test the research hypothesis that process patterns improve the quality of software projects. This chapter presents and discusses the details of the design and conduct of the controlled experiment.

**Chapter 6 - Measurement Process**

The experimental research methodology includes a measurement process that defines the process of defining measurement goals to be achieved and developing metrics that help achieve them. The chapter presents a specific measurement process for this study, through which a number of software attributes are measured using a number of defined metrics. The measurement process also defines the data collection and storage procedures.

**Chapter 7 - Results**

The results of the controlled experiment are presented in this chapter. The detailed results of each of the defined metrics are individually presented. The results of tutor marks used in the experiment will also be presented. The results show whether each metric or tutor mark indicate significant improvements as a result of using process patterns.

**Chapter 8 –Analysis**

This chapter present an analysis of the experiment results. It presents an overall and concise representation of the results. It further discusses the results of each metric in terms of its effect and meaning on any particular software attribute.

**Chapter 9 - Conclusion**

In this chapter a brief discussion of the research's contributions and a summary of the achieved results are presented. The chapter discusses the overall effect of this research in the field of software engineering. Constraints and limitations of the study are also discussed. There will be a discussion and introduction of possible areas of related research for future work.

---

## Chapter 2 Software Engineering Patterns

---

### 2.1 Introduction

In this chapter the concept and application of patterns in software engineering is critically discussed and the main related works are reviewed. There will be a discussion of the major concepts and components of software patterns as well as the general issues surrounding the use of patterns in software engineering.

In the 'software engineering and patterns' section, there is a discussion of the main characteristics of the pattern concept, its emergence and utilisation in the field of software engineering. The 'pattern discussion' section addresses the topical issues on software patterns such as pattern mining and pattern formalisation. 'Patterns in software program design' discusses the patterns concerned with the product aspect of software development (e.g. design patterns). This is followed by the 'patterns in software development processes' where patterns concerned with development process activities (i.e. process patterns) are discussed and reviewed. In the final section, the pattern concept is discussed with respect to task analysis utilities. The section covers the relationship between task analysis and patterns and looks into task analysis with a view to determining whether it can be used in developing and applying patterns.

### 2.2 Software Engineering and Patterns

Software patterns are becoming an important and integral part of software engineering and software development and numerous articles and books have been published on the subject [PLoP 1994 to 2007]. When in an engineering discipline it is possible to name, study, and apply patterns relevant to that domain, it is an indication of the maturity of the discipline [Booch 2008]. In this section, the background to the application of patterns in software engineering is discussed.

#### 2.2.1 How Patterns Entered Software Engineering

Following the research works of Alexander and his team on town building and architecture and the subsequent publication of the work in a number of books [Alexander 1977, 1979, and 1988], the concept of pattern was explored by computer science researchers and software engineers with a view to determining its applicability in software engineering. They realised that the pattern concept might indeed be applicable in solving some of the design problems in software engineering, due to similarities between the construction of architectural entities and software applications. Cunningham and Beck [1987] were amongst the first researchers who introduced the concept of patterns in software engineering with the publication of a paper on Smalltalk interfaces. This was followed by the publication of some patterns for C++ produced by Coplien [1991]. Following the publication of his thesis, Gamma continued his work in software designs and together with three other experts, known as the 'Gang of Four' (GoF), produced a design pattern book [Gamma et al. 1995] which is widely accepted as the authoritative reference book on patterns in software design and development. The publication and subsequent acceptance of this book within the software community established the pattern concept in software engineering. The pattern concept contributed towards the notion of agile process in software development methodologies such as Extreme Programming (founded by Beck [2000] who was one of the pioneers of the software pattern movement).

The popularity of software patterns initiated a forum and a conference named PLoP (Pattern Language of Programming Design) organised by a group of pattern pioneers named the Hillside Group, in 1994. The group set up the conference and devised a set of protocols for the conference to suit the pattern properties. Protocols such as writer's workshop instead of presentations, disregard of originality, and focus on practicability was and still is what set it aside from other conferences [Buschmann 1996]. Apart from the main PLoP conference, held in the USA, there are other worldwide PLoP conference (i.e. EuroPLoP, KoalaPLoP, ChiliPlop, MansorPLoP, SugarLoafPLoP, VikingPLoP) which actively accept and publish papers on patterns. International Journals such as IEEE and ACM also publish patterns related papers from time to time.

For over a decade now software patterns have influenced the way software is designed and developed and have become part of software development mainstream [Kircher and Völter 2007] [Buschmann et al. 2007]. While



traditionally patterns were generally used in Object-Oriented development, they are now influencing aspect-oriented and model-driven software development. Design patterns are included in many CASE (Computer-Aided Software Engineering) tools, which have encouraged the automated inclusion of an implementation of patterns in code. Although it should be noted that patterns should not be ideally used in this way as CASE tools do not understand design and therefore, blind inclusion of patterns in design might prove to be damaging [Kircher and Völter 2007]. Some studies have indicated that the majority of software development organisations studied used patterns in their development practice [Manolescu et al. 2007] [Estabraghy and Dalcher 2007b]. The popularity and application of design patterns is such that many popular Integrated Development Environments (IBM, Sun) now include tools for utilising design patterns by default. Furthermore, numerous books, scientific papers, and articles are published on software patterns (e.g. PLOP Conferences). It appears that the pattern concept has now become one of the most widely applied and important ideas in software architecture and design and is becoming a part of the software development practice. However, such success has so far been limited to a single type of software pattern (i.e. design patterns, in particular Object-Oriented design patterns) partly due to the simplicity in the adoption and application of pattern concept in software design. Given the success of patterns in software design, it is likely that other types of software patterns will also find success in implementation in due course, as patterns mature and become better understood within the software development industry.

### 2.2.2 Software Pattern Definition

There are various definitions given for *pattern* by different authors depending on their views of the concept. While many simply define a pattern as 'a proven solution to a problem in a context', such a definition lacks the recurrence aspect and appears too simplified and incomplete. Each pattern is both a statement in a pattern language and a configuration in a program [Gabriel 1996b] that conveys the essence of a proven solution to a recurring problem within a certain context within competing forces [Appleton 2000]. Noble [2002] describes a pattern as a sign where the signifier is the pattern's solution and the signified is the pattern's intent (i.e. its problem, context, known uses, and rationale). Riehle and Zullighoven [1996] define pattern as the abstraction from a concrete form, which keeps recurring in specific non-arbitrary contexts. Adolph et al. [2002] consider patterns as strategies, stating that, as such, they help people thread their way through complex situations. Alexander [1979] describes a pattern as a rule which describes what you have to do to generate the entity which it defines. Alexander [1979] also provides many statements such as 'Each pattern is a three-part rule, which expresses a relation between a certain context, a problem, and a solution'. The fact is however that the pattern concept is too profound and multidimensional to be able to be properly defined in one or two sentences and therefore the definitions differ, depending on which aspect of the pattern is the focus of interest. One really needs to read Alexander's books [1977, 1979] carefully to fully understand the philosophy and concept of patterns. However, for practical reasons it is generally agreed that essence of a pattern is a problem and a solution where the problem is elaborated in terms of its context and applicable forces.

### 2.2.3 Pattern Elements and Types

Each pattern is described in terms of a number of elements. There have been many proposed formats for patterns and the number of elements in a pattern has not been traditionally fixed in order to render patterns flexible. Consequently, patterns have been published containing a range of between 3 to 13 elements. There are however a few elements, which have become generally accepted to be included in the pattern template. The Figure 2-1 illustrates the way these elements are interlinked in providing the pattern solution. These elements are:

**Name:** It is a word or short, meaningful phrase to describe the pattern.

**Problem:** States the specific problem to be solved.

**Context:** States when to apply the pattern.

**Forces:** Presents the considerations that must be weighted to reach the best solution.

**Solution:** Describe the elements that make up the design, their relationships, responsibilities and collaborations, but no implementation.

**Resulting Context:** A description of the state of the world after the pattern has been applied. Potential users of the pattern can study this section to weigh the costs and benefits.

**Rationale:** Explains the knowledge source and the key factors that makes the pattern useful and effective

**Relating Patterns:** State other related patterns. They may solve the same problem or that are situated in the same pattern hierarchy

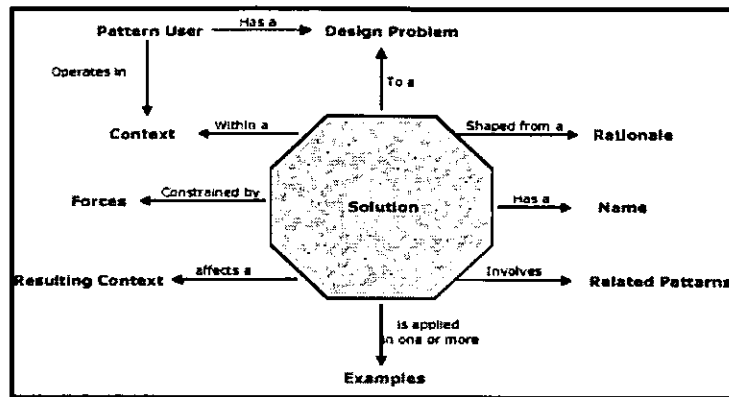


Figure 2-1 Elements of a pattern (www.hillside.net)

While there are many types of software patterns, they can be broadly categorised into the following three types:

- *Analysis pattern*: concerns analysis models that address conceptual structures of business processes rather than actual software implementations
- *Design patterns*: concerned with actual software (code and design) implementations
- *Process/Organisational Patterns*: Concerned with organisational and development processes of software development

Design patterns and process patterns will be further discussed in Sections 2.4 and 2.5 respectively.

## 2.2.4 What Patterns Are, and What They Are Not

There are some misconceptions and misunderstandings, within the software community, as to what constitutes a pattern. Patterns in software design are used to capture knowledge and proven solutions to design problems and provide a mechanism for reusing the knowledge of experienced practitioners. They attempt to provide a proven solution for problems that keep appearing repeatedly. Software patterns capture important practices of existing methods and practices, and are not concrete software components, systems or design methods. They focus more on the human activities of software development than on operations that can be blindly automated, and encourage human intelligence that separates people from computers [Coplien 1996]. However, patterns are not a "silver bullet" [Rising 1999] and do not address all the reuse and other issues in software engineering. The value of their application is largely dependent on both the environment in which they are implemented and the skills and expertise of the pattern implementers. Therefore, the fact that patterns are implemented in an application, does not mean that they have been appropriately and properly implemented. In a pattern, the precise description of the problem, and the context in which it is the best solution, is as important as the solution offered. Accordingly, a full understanding of the problem and the context is crucial in choosing appropriate patterns to apply to specific problems.

It should be further noted that design patterns are not design processes. A design process, typically takes place during the design stage of software construction and results in a concrete system providing solution to a specific problem. A design pattern, however, reflects a generic aspect rather than a particular system. An unlimited number of concrete systems or programs may conform to a single design pattern and, therefore, programs most often constitute instances of design patterns.

## 2.2.5 Disregard for Originality

The concept of pattern relies heavily on proven solutions, and therefore the pattern community seek to capture proven ideas and solutions in patterns. This is somewhat in contrast to the normal research and development policies where innovation, invention and novelty are valued and rewarded. Therefore, by definition a solution that is new and untried cannot be represented in a pattern – a solution can only become a pattern when it is applied empirically and is proven to work in at least three different situations. Although the pattern community has "complete disregard for originality" [Gabriel 1996b], novel solutions are not completely discarded as existing patterns can be applied in novel ways to create novel designs and solutions. Furthermore, there is a direct relationship between new solutions/concepts and patterns as today's new ideas and proven solutions may become tomorrow's patterns. It is the researchers and practitioners endeavour to detect and extract workable

solutions in pattern formats, while discarding solutions that are new and novel. This is to ensure that patterns document proven and workable solutions.

### 2.2.6 Characteristics of Patterns

For a solution to be a pattern, it has to have the following characteristics:

1. Contain the elements and structure of a pattern (i.e. pattern template)
2. Has to be recurring phenomena

It has been generally accepted within the pattern community that a solution has to have recurred at least *three times* in different situations and by people other than the pattern author, in order for it to be a pattern [Hillside Group]. This is referred to as the 'rule of three'. If the pattern does not fully comply with this rule, it is called a *proto-pattern* (a confirmed pattern in waiting).

Patterns by nature can be 'good' or 'bad'. There are practices that are widespread and recurring, but the solutions they provide are wrong. Such 'bad' (inappropriate) practices are captured in patterns that are called *anti-patterns* (also referred to as re-factoring pattern). In other words, anti-patterns are negative solutions that present more problems than they address [Brown 1998]. There is however another more positive definition of anti-patterns in which they contain both the correct and incorrect solution to the problem. In that context, the problem is depicted as a commonly occurring mistake [Laplante 2006]. There exists an interesting relationship between pattern and anti-patterns - patterns can often evolve into an anti-pattern. A popular pattern such as procedural programming can be the popular paradigm of one era and fall out of favour in the next as its consequences are better understood. Therefore while procedural programming would have been a 'good' pattern a couple of decades ago, it can now be considered as an anti-pattern, partly because it does not provide the necessary encapsulation mechanisms. Another example of anti-patterns in software project management is the sequential development process (i.e. the waterfall process) which in the past would have been a pattern, but is now considered an anti-pattern for many application types, as software requirements are now considered moving targets [Brown 2000]. The term anti-pattern however has a negative connotation and it would be better to use the term 're-factoring patterns' instead. For an example of an anti-pattern (see Appendix B. Patterns).

It can be argued that the broad definition of pattern is too general, resulting in the generation of many patterns that are too insignificant to be true patterns. Some within the pattern community believe that a pattern should have characteristics beyond the attributes given above and Winn and Calder [2002] suggest nine such characteristics. According to this characterisation, a pattern should:

1. Imply an artefact
2. Bridge many levels of abstraction
3. Be both functional and non-functional
4. Be manifest in a solution
5. Capture system hot spots
6. Be part of a language
7. Be validated by use
8. Be grounded in a domain
9. Capture a big idea

These characteristics however may be either too restrictive in some cases and/or rather vague (open to different interpretations - e.g. 'big idea') in others. While in principle compliance to these criteria would render patterns more useful, it would lead to the generation of fewer patterns. While it is neither sensible nor feasible to generate patterns for any insignificant problem, one has to be careful not to restrict the pattern requirements to the level that would cause the generation of only patterns that are thought to be significant, and miss or ignore a vast number of other potential patterns.

Patterns manifest their true power when they are part of a pattern language, rather than stand-alone, where they are closely related and collaborate in solving a particular problem. Patterns in a pattern language are often so closely interwoven that they cannot exist in isolation [Buschmann 2007]. Coplien [1995] produced one of the first software pattern languages in the field telecommunication systems. However, unfortunately many of the most popular pattern collections produced [Gamma et al. 1995] [Buschmann 1996] are stand-alone patterns. One way of enhancing the applicability and scope of stand-alone patterns, in order to apply to more complex problems, is to incorporate them into a *sequence pattern*. In effect, pattern sequences provide a way of

combining patterns to solve wider design problems than can be solved by individual patterns [Siddle 2007]. An example of a sequence pattern is depicted in Table 2-1. The example demonstrates how a number of patterns are executed in a sequence to achieve the required solution.


	Pattern	Functionality in the Architecture
	Explicit Interface	Adds explicitly defined service interface
	Encapsulated Context Object	Introduces object representing service discovery context
	Decoupled Context Interface	Decouples service from context implementation by introducing service discovery interface
	Proxy	Adds client-side object, implements explicit interface, and encapsulates remote communication
	Invoker	Adds service side object, receives service invocations, and invokes explicit service interface
	Lookup	Provides ability for service to obtain specific proxy for remote service implementing explicit interface.

Table 2-1 Pattern sequence to add support for service interfaces [Siddle 2007]

### 2.2.7 Software Patterns and Pattern Principles

For all the popularity of patterns in software engineering, Alexander [1999], the pattern concept founder, does not seem to agree that his concept of pattern theory is fully understood and applied in software engineering. He believes that the main principle and strength of patterns, which is in helping create components that are cohesive and generative, has not been implemented in software engineering [ibid].

As well as the technical aspect, software patterns should emphasise the importance of the social and moral issues in software applications and generally aim to improve human life through better software. This is something which is, by large, missing in many types of software patterns (such as design patterns) because they only address the technical aspect of software development. The exceptions are those that concern Human Computer Interaction and usability, which include the human factor [Graham 2003]. One of the main and important published works in the field of design patterns is the work of Gamma et al. [1995], which not only set the standards for design patterns but has also had a major influence on the adoption and popularity of patterns in software engineering. However, one can argue that these patterns lack the key principles of patterns as proposed and defined by Alexander [1977] (see Section 2.4). Based on the progress made so far and the forward trend, one can predict that the role of software patterns will continue to diversify and improve future software engineering tools and methodologies. However, it will probably be a long time before the software engineering discipline will be able to fully implement and utilise the concept and principles of patterns. For that, software engineering needs to become more mature. This view is supported by Gabriel [1996] who writes: “Software engineering is not yet engineering and won’t be, cannot be, for decades, if not centuries, because we cannot yet recognise the important, repeatable parts. When we do we will have patterns – recurrences, predictability”.

The concept of software pattern needs to be given a fresh outlook and emphasis to concentrate on the human, harmony and aesthetic aspect of patterns, rather than solely on its technical utility of capturing and recording software design and experience. The technical benefits that so far have been attributed to the use of patterns should not undermine its other essential utilities. Based on the theory and concept of patterns as proposed by [Alexander 1977,1979], the pattern concept has much more fundamental value and potential and, so far, the rather superficial and simplistic aspects (i.e. capturing design knowledge) are mostly being utilised in software engineering through works such as [Gamma et al. 1995] and [Buschmann et al. 1996, 2007]. While patterns are useful in capturing design knowledge, that should not be seen as their only contribution in software engineering. Patterns’ other important contribution in software engineering should be to help generate systems that fit perfectly in the environment in which they operate by interlinking patterns that are individually and collectively geared to create harmonious systems. Such outlook on the pattern concept and their applications will result in systems that will ultimately perform better. It is time software engineers moved on, from considering patterns as disparate solution packages, to understand and leverage the value and strength of patterns as a collection of interlinked and cohesive solutions for generating perfectly adapted systems. It is only then that software engineering will begin to benefit considerably by the pattern concept.

## 2.2.8 Software Pattern Usage in Industry

Based on a review of the current literature on software engineering in this research, apart from few works such as [Beck et al. 1996] [Manolescu et al. 2007], there has been little industrial level assessment of software patterns. Consequently, part of this research included an investigation through a survey research, focusing on the impact of patterns in the software development industry in the UK, discussed in Chapter 3, which showed that the majority of respondents used software patterns. A recent work by [Manolescu et al. 2007] indicates that there was an increasing gap between expert pattern user/developers and the average software developer in terms of utilising and developing patterns. The study however indicated that the majority of organisations investigated used and attempted to develop software patterns. While the paper [ibid] reports on the popularity and usage rates of various types of patterns in software development industry, the paper has not carried out statistical analysis of the survey results, perhaps due to the insufficient number of respondents. In addition, the paper states that over 70 organisations responded to their survey but failed to elaborate on the exact number or the type and size of respondent organisations. The results and conclusions of the paper however can be seen as some evidence of pattern usage and a snapshot of the status of software pattern implementations in industry.

Beck et al. [1996] made a study of some large software development organisation with respect to their usage of design patterns. Their study indicated some positive results on the utility of design patterns as depicted in Table 2-2.

Patterns	FCS	AT&T	Motorola	BNR	Siemens	IBM
Are a good communications medium	√	√	√	√	√	√
Are extracted from working designs	√	√	√	√	√	√
Capture design essentials	√	√	√	√	√	√
Enable sharing of best practice	√	√	√	√	√	√
Are not necessarily object oriented		√	√	√	√	
Should be introduced through mentoring	√				√	√
Are difficult time consuming to write			√	√	√	
Require practice to write					√	√

Table 2-2 Results summary [Beck et al. 1996]

Based on their study, the paper made the following three, rather generalised, conclusions on design patterns: 1) provide 'shorthand' for communicating complex concepts effectively between designers, 2) can be used to record and encourage the reuse of "best practices", and 3) capture the essential parts of a design in compact form. Researchers and authors have since used these claims of the beneficial effect of design patterns as evidence of the utility of design patterns. However, one should keep in mind that this was a small study based on a small number of samples – too small to be able to generalise the results or make generalised conclusions. Nonetheless, this paper reports on one of the first such studies to evaluate the utility of software patterns and, while the conclusions achieved cannot be statistically generalised, they prompted other studies [Prechelt 2001, 2002] to evaluate patterns and examine the validity of the conclusions reported in this paper.

In the next section various important and topical issues and aspects of patterns is discussed.

## 2.3 Pattern Discussion

There are many issues on software patterns, which have generated many discussions and viewpoints within the software engineering and pattern communities. In this section, these points and issues are raised and discussed.

### 2.3.1 Pattern Mining

There are a large number of 'proven solutions to specific problems' used by software practitioners. It is important that such problem/solution pairs, which are proven to work and are useful, should be extracted and written up in pattern format in a database repository, so that they can be utilised by others. Rising [1998] proposed a number of ways of mining such experience and knowledge and reproducing them in pattern formats. These techniques include interviewing, workshops, meetings, classes, books and articles. Pattern mining is already happening and, apart from patterns presented in published books and papers, there are currently a number of pattern repositories (e.g. Grady Booch's Handbook of software architecture ([www.booch.com/architecture](http://www.booch.com/architecture)), Portland Pattern Repository [<http://c2.com/ppr>]). However, while there is one that contains over 2000 patterns [Booch 2008], most repositories contain very few patterns and many proposed and published patterns are still scattered in various papers, articles and conference proceedings. While in engineering

disciplines, such as civil, mechanical, and electrical engineering, fundamental elements of the common architecture styles in works can be exposed and compared, such actions are extremely difficult if not impractical in software-intensive systems due to a lack of architectural reference [ibid]. Many potential pattern users have stated that they have had difficulty in finding appropriate patterns applicable to particular problems [Manolescu et al. 2007]. There should be a more concerted effort by the pattern community to record such published patterns in a repository, to make search and extractions of such patterns easier. However, having a repository where articles on patterns are stored will not provide the desired outcome. Rather than simply storing pattern papers and articles in a repository, the patterns in such papers should be edited, categorised and stored in a database that is based on key indices. Provision of such databases would render searching for the right patterns systematic and more fruitful.

### 2.3.2 Can Patterns be Harmful?

Patterns can be harmful if they are inappropriately used and implemented. Patterns, in particular design patterns, are often difficult to understand by inexperienced software engineers due to the complexity of the problem/solution they describe, as well as the detailed and abstract way in which they are presented. Jalil and Noah [2007] found novice programmers had difficulty in systematically choosing and applying design patterns. It is essential that the problem to be addressed is fully understood with respect to its context and the forces acting on it, before using a pattern to resolve it. There are two situations where patterns can be harmful:

1. A lack of full understanding of the problem domain
2. Misunderstanding patterns

Patterns can therefore be misapplied, forced to fit, or overused in which case their application would be harmful rather than beneficial. There are many instances in the literature [Shalloway 2003] in which incomplete comprehension of the problem domain, or misunderstanding of patterns, may have caused the application of inappropriate patterns to the problem. Wendorff [2001] reports on a large commercial project where the uncontrolled use of patterns contributed to severe maintenance problems, which required substantial re-engineering effort to be put right.

It is important that the context in which a pattern is applicable is well understood before attempting to use it. As well as understanding and considering the applicable forces, one must also understand and take into account the consequences of applying a pattern to a problem. A misunderstanding of these factors could result in a situation where an application of a pattern could be harmful. For instance, the 'code ownership' process pattern requires individual developers to own specific code that only they should modify. However, for this pattern to work the architecture needs to be interface-based and, therefore, application of this pattern to other types of non-interfaced architecture could prove harmful.

### 2.3.3 Do Software Patterns Work?

There have been many claims in the literature on the advantages and usefulness of software patterns [Buschmann et al. 2007] [Gamma et al. 1995] [Beck 1996] [Gueheneuc 2001] [Rising 1998] [Larman 2002]. These include the following:

- Provide a common vocabulary for designers to communicate documents and explore design alternatives, and therefore improve communication between designers and maintainers.
- Offer "best practices" solution to common problems.
- Capture/record the experiences of expert designers.
- Help beginners to learn by example expert solutions.
- Make the system adopting patterns more flexible and easier to understand
- Facilitate reusing, exporting and importing design ideas
- Reduce the number of defects

While such claims may sound plausible, there has not yet been enough empirical research to verify them. It is one of the aims of the research undertaken in this project to provide some evidence regarding the utility and benefits of software patterns. There have been some empirical studies showing that patterns enhance communication between designers by providing a shared vocabulary [Unger and Tichy 2000]. However, there have also been some studies indicating that software patterns are disadvantageous for making software complex and more error-prone [Bieman et al. 2003]. There is therefore a need for further empirical work to evaluate software pattern potential and utility in software engineering. It has been the main aim of this research to provide

some scientific evidence, through an empirical study, on the utility and benefits of software patterns. This study has provided evidence that the application of process patterns improve the quality of a software development project in terms of a number of attributes (see Chapter 7 and Chapter 8).

One, however, should be cautious in concluding that patterns work by the results of a few studies. Many empirical studies have limited scopes and their conclusions may only be valid for the environment in which they are conducted. The question of whether software patterns work can only be convincingly answered when there has been much more research carried out on various types and aspects of software patterns at both academic and industrial levels. Furthermore, even if it is proven in theory that patterns work, the question remains whether they would work in practice as implemented in the software industry. One difficulty in pattern implementation and usage is the need for complete understanding of the pattern, as well as the problem they are aimed to solve. Therefore, patterns may not work in practice in some situations because they are implemented incorrectly. In such situations, it is not the patterns that do not work, but their implementation in a wrong and invalid context. It is therefore essential that more empirical research, such as this, be conducted to enhance our understanding of software patterns and their strength and weaknesses in helping to develop high quality software.

### 2.3.4 Should Patterns Be Formalised?

There is an on going argument on pattern formalisation within the pattern community. The exponents argue that pattern formalisation is necessary and provides a number of important advantages and benefits [Eden 1999] [Dittmann et al. 2002] [Bayley and Zhu 2007]. Formalisation of patterns would enhance the clarity and accuracy of patterns, and the relationships between them, by imposing logical and mathematical constraints. It would further help in developing a more formalised pattern validation process, as well as making it easier to develop comprehensive patterns tool support to automatically and systematically detect and implement patterns. There are already many proposed techniques and tools for detecting GoF's design patterns in a program, such as [Tsantalis 2006], which is based on calculating the similarity between a particular design pattern and the target program, in terms of the structural relationships between classes. A technique to formalise process patterns has been proposed by Dittmann et al. [2002] through introducing what they call a *Process Pattern Description Language* (PPDL). The proposed PPDL uses UML notations to depict the necessary process tasks in solution element of the pattern. While useful, the proposed technique only involves the addition of UML notations to the solution element of the pattern and does not truly formalise process patterns. Furthermore, the proposed PPDL will only be applicable to a restricted number of processes. In general, the formalisation of patterns may be disadvantageous for the following reasons:

- Patterns are intangible, elusive, and hence beyond the scope of mathematical expression
- There is no fixed element in patterns, and everything can be changed about them. In other words, if the basic structure were fixed then it would not be a pattern any more. [Coplien 1996]
- Patterns are abstractions, or generalisations, and therefore are not appropriate to be expressed in mathematical terms
- Formalising the solution makes it harder to grasp the key ideas of the pattern, programmers need concrete information that they can understand, not an impressive formula." [Buschmann et. al 1996]
- Formalisation damages the human factor notion of patterns rendering it more automated and less human oriented

While there are benefits to formalising some types of patterns, the disadvantages that would ensue outweigh the benefits. Patterns, therefore, should keep to their original principal of being flexible and abstract, and remain unformalised and free from constraints that such a measure could impose. Formalisation may prove beneficial for some types of software patterns in the future when both the software pattern concept and the domain in which they relate are better understood and established. Currently, however, formalisation of patterns is neither practical for all pattern types, nor is it feasible or beneficial. Research should be encouraged to develop tools for pattern detection, composition, and application, accounting for the abstract and flexible nature of patterns.

Patterns are applied to both product and process aspects of software development. In the following section patterns related to the product aspect of software development is discussed.

## 2.4 Patterns in Software Design

As discussed above, the first applications of patterns in computing science was in the field of software design (i.e. design patterns). A design pattern names and explains a general design that addresses a recurring design issue in a software application [Schmidt 2000]. It describes the problem and the solution, as well as when to

apply the solution and what would be the consequences of its application. It also offers implementation hints and examples. The solution is a general arrangement of objects and classes and is customised and implemented to solve the problem in that particular context.

Amongst the most prominent work on design pattern, is the work of GoF in the book 'Design Patterns, Elements of object oriented Software' [Gamma et al. 1995]. The book introduces 23 object-oriented design patterns in all, which are divided into three distinct categories of object creation, object structure, and object behaviour. Each pattern within these categories presents a solution to a common recurring problem in software development. Although there have been many more design patterns published since, these design patterns (known as GoF Patterns) are the most widely known and used, and present solutions to the most common object-oriented design problems [Manolescu et al. 2007]. The impact of GoF's work [Gamma et al. 1995] has been such that often the phrase 'design patterns' refers to the patterns introduced in this book. However, partly due to the influence of this book, most designers wrongly think that design patterns are only applicable in object-oriented designs [Sommerville 2007]. Design patterns as a way of encapsulating experience are in fact equally applicable to all software design approaches.

For the format and structure of the design patterns, GoF use a format that is more detailed than that which Alexander used in 'A Pattern Language', containing 13 elements as depicted in Table 2-3. The comprehensive set of elements can be considered as strength of the work in fully documenting all aspects of the introduced patterns. However, it can be argued that it would have been perhaps more appropriate to the flexibility and abstract nature of the pattern concept had there been fewer elements. That would not indicate a loss of content, but a rearrangement to repackage the conveyed information in fewer elements. For example, some of these elements (e.g. Motivation, Applicability, Consequences, Collaborations, and Participants) could be integrated into more conventional pattern elements such as *Forces* and *Context* (2.2.3). It has to be acknowledged however that software design and architecture is often complex and intrinsic and, therefore, one has to accept the extra complexity in the pattern formats (e.g. no. of pattern elements) to enable full and unambiguous description of the pattern at the expense of simplicity.

Element	Description
Pattern Name	What is the pattern called?
Intent	What problem does this pattern solve?
Also known as	What are other names for this pattern?
Motivation	What is an example scenario for applying this pattern
Applicability	When does this pattern apply?
Structure	What are the class hierarchy diagrams for the objects in this pattern?
Participants	What are the objects that participate in this pattern?
Collaborations	How do these objects interoperate?
Consequences	What are the trade-offs of using these
Implementation	Which techniques or issues arise in applying this
Sample Code	What is the example of the pattern in source code?
Known uses	What are some examples of real system using this pattern?
Related patterns	What other patterns, from this pattern collection, are related to this pattern?

Table 2-3 GoF's design pattern elements [Gamma et al. 1995]

Gamma et al. [1995] claim that their patterns have all been fully tested and proven to work. In fact, the last element of the pattern format is 'Known Uses', which contains the details of where the patterns were applied. Furthermore, the patterns were tested on a specifically developed application named ET++. Although there are a number of patterns (23 in all) in this book, they fail to be coherent enough to form a pattern language. At one level, they can be considered as just a library of C++ code templates. For these to comply with the true principles of the pattern concept they needed to be much more coherent. They are isolated and rather disjointed, and do not interrelate as they would have in a pattern language. Gamma et al. [1995] admit that the design patterns introduced in the book do not completely conform to the definition of a pattern language, arguing that it had not been an aim or objective of the work to represent a pattern language. However, they produced 23 of the most popular design patterns, which are widely used by software practitioners, about which many books and articles have been published. The introduced patterns have been widely discussed and studied by both researchers and practitioners [Beck et al. 1996] [Bieman et al. 2003] [Prechelt et al. 2002].

Apart from GoF's book [Gamma et al. 1995], another important publication on design patterns is a book by Buschmann et al. [1996] which defines several well-known design and architectural patterns, such as the Proxy,



Whole-Part, Master-Slave, and Broker, presented in three hierarchical levels. At the highest level are the architectural patterns, followed by the design patterns, and finally idioms. It introduces eight well-known patterns where each pattern is described and discussed to a deeper level and more clearly than those introduced by Gamma et al. [1995]. Patterns introduced in this book were amongst the first published software architectural patterns, presenting some established architectural solutions in pattern formats. However, one weakness of the book, which could prove problematic to the novice pattern user, is that it does not clarify, through examples or otherwise, where and how the patterns should be used. Further volumes of this book have been periodically published, each introducing many established software architectural patterns for different domains and technologies.

There have been many claims in the literature regarding the positive effect of design patterns. It is claimed that designing an application with the proper use of design patterns would reduce the number of defects [Gueheneuc and Albin 2001]. There have also been other claims that using design patterns provides additional flexibility and easier understanding of the design [Rising 1998] [Buschmann et al. 1996, 2007] [Larman 2002]. There have also been some studies reporting negative effects of design patterns. Bieman et al. [2003] studied five systems (three proprietary systems and two open source systems) to identify the observable effects of the use of design patterns on the changes that occur to the systems as they evolve. In this paper, a number of design patterns were used in the early versions of some applications and the errors contained in the later versions were compared to those applications that did not use any design patterns. The study indicated that, in four out of the five systems studied, the application of design patterns made the systems more error prone. The use of design patterns appears to have caused a higher number of errors, which is contrary to expectation. A number of reasons, such as incorrect application of patterns, as well as a lack of understanding of the applied patterns by the involved programmers, could have been the cause. The inconsistent and at times contradictory results reported in the literature, on the very few studies that have been carried out on pattern utility, necessitate further studies. Unfortunately, design patterns have been associated with a degree of hype [Hillside Group] in the industry, and often it is taken for granted by many that GoF's design patterns are useful whenever and wherever they can be applied. The fact is however that the value and utility of patterns can only be evaluated and judged through scientific studies that exclude hype and bias. Unfortunately, so far only very few such studies are reported in the literature and there is therefore a need for many more empirical studies to be conducted to evaluate software patterns.

Design patterns are, however, concerned with product (which is the result of the design), not the process of designing. In the following section the process aspect of software development, and the application of patterns within them, is discussed.

## 2.5 Patterns in the Software Development Process

Development of a software application requires a development process that is designed to orchestrate and control the activities and tasks within the software development project. The quality of software products relies significantly on the quality of the process used to design, develop, deploy, and maintain them [Fuggetta 1998]. Patterns that deal with development process activities are referred to as *process patterns*. They are the main topic of this research, in which their utility and effect in software development is empirically examined through experimentation.

Process patterns attempt to provide a mechanism for communicating approaches to development that have proven to be effective in practice. According to Ambler [1998], they are the reusable building blocks from which organisations may tailor a mature software process. Process patterns are similar to design patterns in principle and concept, except that they exist in the process domain. There are a number of works on this topic including a two-volume book covering patterns in all major phases of a complete development lifecycle [Ambler 1998, 1999]. There are various definitions for process patterns. Coplien [1995] defines process patterns as "Patterns of activity within an organisation and its projects". Storrle [2003] simply states that a process pattern describes a piece of a process. Ambler [1998] defines process patterns as "Patterns which describe a proven solution, successful approach and /or series of actions for developing software."

A major contribution to the concept of process patterns is Coplien's paper [Coplien 1995], which was presented at the first PLoP (Pattern Language of Programming) conference. This paper was the result of a three-year research at the AT&T, which investigated the software organisational structure as well as the development practices. Based on this study, the paper introduces 43 patterns, which it claims improve organisations' development processes. These patterns follow a standard template consisting of five elements: Problem, Context, Forces, Solutions, Resulting Context, and Rationale. Most of the patterns introduced in this paper are short and concise. An example of a simplified process pattern introduced by Coplien [1995] is shown in Table 2-4.

<b>Name</b>	Prototype
<b>Problem</b>	Early acquired requirements are difficult to validate without testing
<b>Context</b>	Trying to gather requirements necessary for test planning
<b>Forces</b>	Requirements are always changing Requirements are usually ambiguous
<b>Solution</b>	Build a prototype, whose purpose is to understand requirements.
<b>Resulting Context</b>	A better assessment of requirements to supplement use cases

Table 2-4 Prototype process pattern

Coplien [1995] argues that an important and significant attribute of the process patterns he introduced is that they are generative patterns (i.e. one pattern can indirectly cause the creation of other patterns or processes [Alexander 1979]). In validating the introduced patterns, Coplien [1995] uses both the 'case study' method and what he calls 'commonsense approach'. The patterns are based on combined empirical observations with a rationale that attempts to explain them.

Whilst it is fully acceptable that the patterns produced in Coplien's paper are indeed the activities practiced at AT&T at the time of this study, it is not always clear whether the introduced patterns are positive or negative in their effect (i.e. whether they are 'good' or 'bad' patterns). While the validity of the patterns can be fully endorsed, as they have appeared to occur in a real life situation, it has not been established whether every pattern introduced will resolve the problem it has been claimed it should. A weakness of the work is its implemented validation method where the author has often relied on his commonsense and rationale as a method of validating the patterns. This implies that the pattern user would have to rely on the author's commonsense and rationale, to a certain extent, to accept that the introduced patterns are useful and reliable. Furthermore, the claimed generative aspect of the produced patterns has not been sufficiently substantiated. It has not been explained (e.g. through examples) how and why the produced patterns would cause the generation of other patterns. A further weakness of the work is that, while the 43 patterns introduced in the paper are related, reference one another, and present a catalogue or system of patterns, they do not form a pattern language. In order to form a pattern language, the patterns needed to be closely linked structurally, and address the whole software development domain.

The strength of this work is in the production of a comprehensive set of process patterns, which were the result of a 3-year case study research. The patterns are succinct and clearly written, presenting all the important pattern elements such as forces and context. The author is also a respected pioneer of the software pattern movement, and has written many books and papers on the subject [Coplien 1991, 1996, 2005]. The paper is widely referenced and is generally accepted as one of the first papers and a key contribution to patterns in organisational and developmental process activities of software development organisations [Ambler 1998]. Many of the patterns stated in this paper are used in the experiment (Appendix A. Experiment Details).

Ambler [1998, 1999] produced a system of process patterns defined hierarchically in terms of the level and scope of the process they describe. These patterns ranged from high-level view of how a specific project phase works, to a more detailed view of a specific task or activity. Three types of process patterns are defined in a hierarchical format:

1. **Task process pattern** depicts the detailed steps to perform a specific task such as the technical review tasks.
2. **Stage process pattern** depicts the steps, which are often performed iteratively in a single project stage. A stage process pattern is presented for each project stage (e.g. model stage)
3. **Phase process pattern** depicts the interactions between the stage process patterns for a single project phase.

The hierarchical structure of these pattern types are depicted in Figure 2-2.

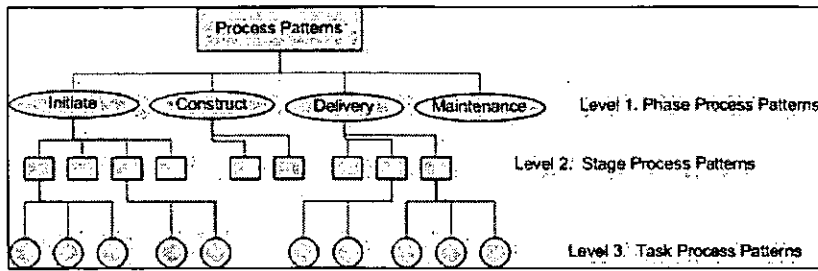


Figure 2-2 Hierarchical structure of process patterns

Figure 2-3 presents an example of a task process pattern and depicts the activities involved in the 'technical review' task process pattern.

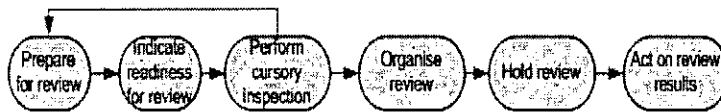


Figure 2-3 Task process pattern for technical reviews

A pattern template, composed of four elements, is used to present the patterns as shown in Table 2-5.

No	Element	Description
1	Name	Name of the process pattern
2	Initial Context	The initial entry condition sets out the condition that has to be satisfied in order to move to the next stage
3	Solution	This element contains the main solutions provided by the pattern.
4	Resulting Context	This element contains the conditions that have to be satisfied in order to complete the pattern

Table 2-5 Elements of process pattern [Ambler 1998]

Ambler's work [1998, 1999] has a number of positive points, which enhances the strength of the presented patterns. While most work on process patterns deal with some aspect of the development process [Whitenack 1994] [Kerth 1995] [Delano 1998], this work covers a complete development lifecycle. The process patterns encompass all major development activities such as requirement analysis, design, development, delivery, and maintenance. The author uses his own experience as well as other documented evidence in writing and validating the patterns. A further strength of the work is the introduction of three types of patterns and the establishment of a hierarchy in which they are presented. The categorisation of the process patterns in three hierarchical levels has helped to make the process patterns concise and easier to understand and implement. Such distinct categories of patterns offer the pattern user the choice of employing any process patterns to the required level. For example, the stage process pattern 'program' describes the activities necessary to accomplish the programming activity. A pattern user may be satisfied with the general guidelines and solutions presented at this level of hierarchy. However, if the user required more details, they could go down a level and study the task process patterns related to lower level activities. Depending on the nature and scope of the problem and the level of solution required, a single or a group of patterns can be employed to solve a single process problem.

The work, however, suffers from some weaknesses. The three hierarchical levels would have benefited from further lower levels with additional detail. The 'task process patterns' are too generic and are, therefore, long. It would have been preferable for the task process patterns to be further divided into two hierarchical levels making the process patterns into a 4-level hierarchy. There is also inconsistency in the pattern format as three different formats are presented. While the format for the phase process pattern and stage process pattern are similar, the pattern format for the task process pattern is different, containing fewer elements. Although one can argue that the different characteristics of the process pattern types necessitate different approaches in their formats, the work would have benefited in terms of its clarity and substance if a consistent pattern format had been used. The presented patterns further suffer from a lack of presentational quality. Some of the patterns are poorly presented and edited and contain typographical and other mistakes (such as unfinished sentences).

Another comprehensive work on process patterns is conducted by D'souza and Wills [1999] who present a set of 54 individual process patterns on object, component and framework development. The process patterns cover three categories of development activities, namely, business modelling, components specification, and component implementation. For each activity category, there are a number of patterns with differing levels of granularity. The patterns for the business modelling activities include *make a business model*, *present business vocabulary and rule*, *involve business experts*, *choose a level of abstraction*, and *generalise and specialise*. The pattern schema used has six elements as shown in Table 2-6.

No	Element	Description
1	Name	Name of the pattern
2	Intent	An account of the rationale of the pattern - its benefits and application area.
3	Context	Circumstances under which the pattern should be applied
4	Consideration	Consideration of the forces involved
5	Strategy	Strategy for presenting the solution
6	Benefit	Advantages of applying the patterns

Table 2-6 Pattern elements in [D'souza and Wills 1999] patterns

One important attribute of this work is its attempt to present the patterns as a pattern language. The language aspect of patterns is an important concept in the pattern theory, which unfortunately, has not been achieved in key works on software patterns. For example, two classical and seminal works on software patterns, [Gamma et al. 1995], and [Buschmann 1996] are both catalogues of patterns rather than pattern languages. In a pattern language, the pattern user is able to select individual patterns to form a sequence of patterns, in a manner that collectively solve a particular non-trivial problem. In such pattern languages, there would be numerous ways in which the patterns can collate and coalesce to solve a problem. This is analogical to the way infinite numbers of sentences can be generated by linking words in a natural language.

Therefore, a key strength of this paper is that the introduced patterns incorporate some pattern language characteristics in the presented patterns. For example, the patterns can be used to plan a route method for the development of new applications, or for reengineering an existing one. Furthermore, the presented patterns cover key tasks in the three components of the development activities mentioned above. The presented patterns are lightweight and present a concise solution for each pattern and, where necessary, UML is used to explain and clarify solutions. The work, however, suffers from some deficiencies. Although it is claimed that patterns are used in case studies presented in the book, it has not been explained which patterns were used, and the circumstances of their application are not stated. The majority of patterns are concentrated on the design aspect and there are few patterns on coding, testing and requirement analysis. Another weakness of the work is the inconsistency in the pattern schema depicted in Table 2-6 where many patterns miss the Context and Benefit elements.

In his PhD research, Storrie [2000] made a comprehensive study of process patterns, in which a number of process patterns have been proposed. He has studied and investigated the architecture centric processes and has anticipated that such processes would be best described by process patterns since they can be applied repeatedly at several levels of abstraction. A non-hierarchical four level classification is used for the proposed process patterns. These classifications are:

1. **Abstraction level:** Techniques and development styles
2. **Phase:** Specification, design, realisation and maintenance
3. **Purpose:** Administration, the construction of proper and quality assurance
4. **Scope:** Project, Component, Style

The template/format used for the pattern description is presented in Table 2-7.

No	Element	Description
1	Title	This is usually identical to the name of the task that is supported by the pattern
2	Synonyms	Other adequate names
3	Classification	Abstraction level, phase, purpose and scope
4	Related Patterns	Relationships to other patterns
5	Intent	An account of the rationale of the pattern
6	Motivation	A scenario that illustrates the applicability conditions and purpose of the pattern.
7	Consequences	Discussion of the advantages and disadvantages of using the pattern
8	Participants	The actors, roles/resources, techniques, activities, tools and document types involved in the process, and their respective roles.
9	Applicability	The prerequisites for applying a pattern, the context where it may be applied.
10	Deliverables	Describes what (parts of) documents are created or changed in which way by applying the pattern
11	Process	The central part of a process pattern is the description of the process fragment itself

Table 2-7 Pattern elements [Storrie 2000] patterns

The strength of this work is the classification, the pattern schema and the set of patterns it contains. The classification proposed has the advantage of being analogical and compatible to the well-established pattern classification proposed in GoF's design patterns [Gamma et al. 1995]. Similarly, while the schema uses fewer elements than those proposed in GoF's design patterns, the elements adopted are suitable and correspond to the nature of the process that the patterns describe. A further strength of the work is the number and scope of process patterns proposed. In total, there are 25 patterns, each of which is well defined in accordance with the devised pattern schema.

The work, however, suffers from many weaknesses, one of which is the lack of pattern name and classification in the pattern schema for the proposed patterns. Although each pattern introduced in this work has the name and classification mentioned at top of the page, it is not included in the pattern schema (template). The absence of these elements in the pattern schema could make the process pattern ambiguous and more difficult to use. It would further complicate the processing, recording and indexing of these patterns in a pattern repository [Portland Pattern Repository]. Another weakness of this work is the quality and completeness of some of the proposed patterns. Some of the patterns seem superficial and lack the broad and full details exemplified in other works [Ambler 1998]. For example, in the pattern 'Analyse Domain', the solution offered for the process is, 'The structure and logic of the application domain should be analysed in the usual ways, resulting in a number of class and activity diagrams in the respective views'. Here it is not clear what the 'usual ways' are, and such patterns or solutions, as offered in this paper, are not substantive and lack elaboration and clarity.

Cary and Carlson [2002] present 25 process patterns on requirements, analysis, and design. The patterns cover issues such as communication, iteration, consistency, incompleteness, and flexibility. The patterns are specifically aimed at framework development and reflect the author's practical experience in framework development. A framework is defined as a set of components working together to address a number of problems in one or more domain. One of the strengths of this work is the number of elements the patterns contain and the comprehensiveness in which each element is presented for all the patterns. The schema use is similar to that of GoF's design patterns and contains 12 elements, which are depicted in the Table 2-8.

No	Element	Description
1	Name	Name of pattern
2	Also known as	Other names for the pattern
3	Intent	What the pattern is about
4	Context	The motivation for the pattern
5	Examples	Examples from case study
6	Problems	A concise statement of the pattern addresses
7	Approaches	Various solution approaches
8	Solution	A concise statement of how the pattern recommends solving the problem
9	When to use	Tradeoffs of the pattern, normally based on the case study
10	Applicability	A concise statement of the tradeoffs
11	Known uses	Places the pattern was applied
12	Related Patterns	Other patterns related to the current one and how they are related

Table 2-8 Pattern elements in Cary and Carlson [2002]

Although the book covers some key themes and aspects of framework development such as communication and flexibility, a weakness of the work is its limited coverage of the extent and depth of development process issues concerned in framework development. However, the issues that are raised are covered well in the process patterns that it presents. A further weakness of the work is that the introduced process patterns are disjointed and disparate and do not form a pattern language, from which a pattern sequence could be generated to solve a detailed and complex problem. While it presents examples of how individual patterns would solve the intended problem, the work does not present scenarios in which a number of patterns could be interlinked to solve a specific problem or achieve a specific goal.

As part of the solution elements, patterns may contain instructions/tasks, which may be required to be carried out sequentially, in parallel or in no particular order. The hierarchical nature of patterns and the tasks/instructions involved are discussed in the following section.

## 2.6 Instructions in Patterns

As part of its solution element, a pattern may contain one or more instructions (tasks) to be carried out sequentially, in parallel or in no particular order. In pattern languages, where a number of related patterns are concerned with a particular problem domain, the relationships between patterns are hierarchical [Alexander 1977]. A pattern language is a structured collection of patterns that build on each other to transform needs and constraints into an architecture [Coplien et al. 2005]. Patterns in a pattern language work together in such a way that a collection of patterns can be selected from the language to provide a solution for a complex problem. Pattern languages have hierarchical structures, where different processes occur on different scales or levels, and connections exist both on the same levels, and across levels [Salingaros 2000].

This hierarchical nature of patterns, in pattern languages or systems, is present in both software and non-software domains. The solution element of a higher-level pattern may contain tasks as part of its solution, each of which could themselves be presented as lower level pattern. For example, 'architectural patterns' are concerned with higher level design and architecture issues, while 'design patterns' and idioms deal with problems at a lower level. This hierarchical nature of the patterns, in a pattern language, is a fundamental concept of pattern. This is illustrated and emphasised by Alexander [1977], where the introduced patterns in the field of architecture, are hierarchically ordered, beginning with the very largest, for regions and towns, then working down through neighbourhoods, clusters of buildings, buildings, rooms and alcoves, ending finally with the fine details of construction. Each pattern is linked to certain larger patterns which come above it in the language, and to certain smaller patterns which come below it. The pattern helps to complete those larger patterns which are above it in the hierarchy, and is itself completed by those smaller patterns which are below it. In a pattern language, therefore, the patterns at higher levels are decomposed into lower level patterns, which address smaller problems. The decomposition of a higher task into smaller, lower level tasks, is an important component of Task Analysis. There therefore appears to be a relationship between patterns and task analysis which will be investigated in the following sections.

While the solution elements of some patterns may present one or more instructions, to be carried out either sequentially or in parallel, that is not a general rule. In order to explore the hierarchical structure and task-based solutions in patterns, various types of software and non-software patterns (i.e. building and architecture patterns, design patterns, and process patterns) are discussed in the following sections. Process patterns in particular are the focus of the discussion.

### 2.6.1 Patterns in Town and Building Architecture

The task based structure of architectural patterns are presented in two typical patterns 'Window Place' and 'Outdoor room' [Alexander 1977] shown in Table 2-9 and Table 2-10. In these patterns the solution is given as one or more tasks to be carried out to achieve the goal and generate the pattern. There could be another pattern associated with each task that would present a decomposition of the given task. For example, the task 'keep the sill low' is dealt with another pattern 'Low Sill (Pattern 222)'. The tasks in these patterns do not need to be carried out sequentially.

<b>Pattern Name</b>	Window Place
<b>Problem</b>	Design of a residential room
<b>Forces</b>	One wants to sit down and be comfortable One is drawn toward the light
<b>Solution</b>	In every room where you spend any length of time during the day, make at least one window into a "window place." <ul style="list-style-type: none"> <li>• Make it low and self-contained if there is room for that – (Alcoves Pattern 179)</li> <li>• keep the sill low – (Low Sill Pattern 222)</li> <li>• put in the exact positions of frames, and mullions, and seats after the window place is framed, according to the view outside – (Built-in Seats Pattern 202), (Natural Doors and Windows Pattern 221)</li> <li>• And set the window deep into the wall to soften light around the edges – (Deep Reveals Pattern 223)</li> </ul>

Table 2-9 Window Place Pattern

<b>Pattern Name</b>	Outdoor room
<b>Problem</b>	Design of an outdoor room
<b>Forces</b>	A garden is the place for lying in the grass, swinging, croquet, growing flowers, throwing a ball for the dog. But there is another way of being outdoors: and it needs are not met by the garden at all.
<b>Solution</b>	Build a place outdoors which has so much enclosure round it that it takes on the feeling of a room, even though it is open to the sky. To do this, define it at the corners with columns, perhaps roof it partially with a trellis or a sliding canvas roof, and create "walls" around it, with fences, sitting walls, screens, hedges, or the exterior walls of the building itself.

Table 2-10 Outdoor room Pattern

Any pattern within the pattern language can be used in a sequence (also referred to as a construct or a language) to generate a solution to a specific problem. For example, the *outdoor room* pattern above can be used in a pattern sequence to build a porch, as shown in Table 2-11.

No.	Pattern Name	Pattern No.
1	Private Terrace on the street	140
2	Sunny Place	161
3	<b>Outdoor Room</b>	163
4	Six-foot Balcony	167
5	Paths and Goals	120
6	Ceiling Height Variety	190
7	Columns at the corners	212
8	Front Door Bench	242
9	Raised flowers	245
10	Different Chairs	251

Table 2-11 An example of a pattern sequence (also referred to as language or construct) for building a porch

These two examples of architectural patterns demonstrated both the hierarchical and task oriented nature of patterns in the field of architecture. The following is a further example of the nature of patterns in the field of software engineering.

## 2.6.2 Patterns in Software Design

The Model-View-Controller architecture pattern [Gamma et al. 1995] [Buschmann et al. 1996] presents a method of decoupling presentations from data in a software application. A simplified version of the pattern is given in Table 2-12 below.

<b>Pattern Name</b>	Model-View-Controller
<b>Problem</b>	How to provide several user interfaces to a set of data
<b>Solution</b>	The following tasks need to be accomplished <ul style="list-style-type: none"> <li>• Encapsulate core data in a model component. (Composite Pattern)</li> <li>• Display information to the user using view component. (Factory Method Pattern), (Decorator Pattern)</li> <li>• Use a controller component to control user interaction with the system (Observer Pattern), (Strategy Pattern)</li> </ul>
<b>Related Patterns</b>	Composite Pattern, Method Pattern, Decorator Pattern, Observer Pattern Strategy Pattern

Table 2-12 Model-View-Controller

The solution presented in this pattern can be seen as a number of tasks, each of which refer to other hierarchically lower level patterns (i.e. design patterns), that help accomplish the task. A further example is a simplified version of the 'Decorator' design pattern [Gamma et al. 1995] shown in Table 2-13 where, in the solution element, the tasks that need to be carried out to accomplish the pattern are stated in a number of tasks. The pattern presents its solution as a number of tasks to be accomplished.

<b>Pattern Name</b>	Decorator
<b>Problem</b>	How to attach additional responsibilities to an object dynamically
<b>Solution</b>	The following implementation issues should be considered: <ul style="list-style-type: none"> <li>• Define the interface for objects that can have responsibilities added to them dynamically</li> <li>• Define an object to which additional responsibilities can be attached</li> <li>• Define a decorator interface to maintain a reference to the main object which defines an interface that conforms to component's interface</li> <li>• Define a decorator object to add responsibilities to the main component</li> </ul>

Table 2-13 Decorator Pattern

The examples above demonstrate the hierarchical and task oriented nature of architectural and design patterns in software designs. In the following section, some examples of process patterns are presented, with a view to illustrating their hierarchical and task oriented nature.

### 2.6.3 Patterns in Development Process

In the same way that there are patterns of different hierarchical levels (i.e. architectural patterns, Design patterns, Idiom patterns) in software engineering, there are also patterns of different hierarchical levels in software development processes that are concerned with distinct levels of abstraction. This is depicted in Figure 2-4 (the three pattern types depicted are further discussed in Section 2.5).

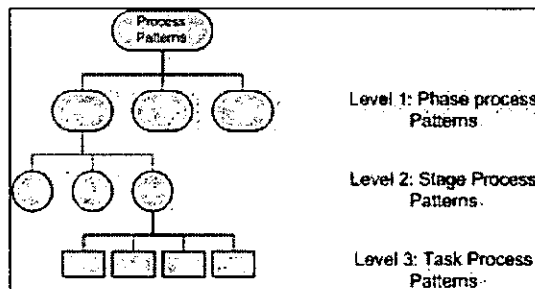


Figure 2-4 Hierarchical structure of process patterns

Similar to the building architecture and software design patterns discussed in the previous sections, process patterns present their solution element as a number of tasks to be accomplished [Ambler 1998]. This is illustrated in a number of examples of process patterns shown in Table 2-14 and Table 2-15.



<b>Pattern Name</b>	Requirement Analysis
<b>Problem</b>	How should work proceed in the requirement analysis phase
<b>Solution</b>	<p>In this phase, the project plan should be put in place and initial requirements are defined. The following parallel activities should be taking place in this phase. Note that all three activities must be taking place at the same time.</p> <ul style="list-style-type: none"> <li>• Defining and validating initial requirements (Pattern <i>Stage_1_1</i>)</li> <li>• Defining the initial project management (Patterns <i>Stage_2_1</i>)</li> <li>• Justifying the project (Pattern <i>Stage_3_1</i>)</li> <li>• Defining the project infrastructure. (Pattern <i>Stage_4_1</i>)</li> </ul>

Table 2-14 Requirement Analysis Pattern

Here each task is linked to a pattern at a lower hierarchy (i.e. a stage pattern) within the pattern language. The pattern user may be satisfied with the solution provided at the *phase* level, or might decide to investigate the related *stage* patterns to get further solutions.

The Table 2-15 depicts a further example of a process pattern, 'Big ball of mud' [Foot and Yoder 1997], which contains a number of tasks. The pattern suggests a number of tasks to be carried out in order to deal with the stated problem of overgrown and tangled code. The tasks are represented as guidelines and suggestions and there is no particular order in which they are to be carried out.

<b>Pattern Name</b>	Big Ball of Mud
<b>Problem</b>	Overgrown, tangled, haphazard spaghetti code is hard to comprehend, repair, or extend
<b>Solution</b>	<ul style="list-style-type: none"> <li>• If you cannot easily make a mess go away, at least cordon it off. This restricts the disorder</li> <li>• To a fixed area, keeps it out of sight, and can set the stage for additional refactoring.</li> <li>• If your code has declined to the point where it is beyond repair, or even comprehension, throw it away it and start over</li> </ul>

Table 2-15 Big Ball of Mud process pattern

Patterns in a pattern language can be combined in a large number of ways to solve a problem [Alexander 1977]. For example, there are numerous ways of building a porch using a number of patterns (one example is shown in Table 2-11). In accordance with this concept, individual patterns of different hierarchical levels in a process pattern language, or system, can be compiled in many different ways to achieve different and specific solutions. For example, a pattern sequence (also referred to as language or construct) for object development from scratch [D'souza and Wills 1999] can be generated, using a number of patterns selected from a pattern language as depicted in Table 2-16. Each pattern contains one or more tasks to be completed to accomplish the solution given by the pattern. Some or all of these patterns could be used in a different sequence to solve a different problem (e.g. re-factoring existing application).

No.	Patterns	Description
1	Make a Business Model	Describe your understanding of the users' concepts and concerns and the vocabulary in which they express them
2	Make a Context Model with Use Cases	Focus on the collaborations between your proposed system and other objects – people, machines, other software systems
3	Construct a System Behaviour Spec	Treating your system as a single object, create a type specification for any system that would meet the requirements
4	Avoid Miracle, Refine the Spec	Define more-detailed actions and attributes as a refinement.
5	Implement Technical Architecture	Define and implement major components of design as a collaboration
6	Basic Design	Take each system action and distribute responsibilities among collaborating internal components.
7	Link and attribute ownership	Extract common components and recast the design in terms of the components
8	Object Locality and link implementation	Decide how the basic design is split among machines, applications and hosts.
9	Optimisation	Perform localised refinements for performance

Table 2-16 A language for object development from scratch

In the above discussions and examples it has been demonstrated that the process pattern are not simply a set or sequence of tight textual instructions, but are dynamic solutions that involve tasks to be performed sequentially, in parallel, or optionally. An aspect of process patterns, which often makes them unsuitable as tight textual instructions, is the necessary human involvement. There are often tasks in patterns and in software development process that involve human judgment and decisions. There is however a fundamental question on the nature of the software process itself, and that is whether software processes are instructions that can be represented in software programs. This is discussed further in the next section.

#### 2.6.4 Software Process and Textual Instructions

There is a school of thought that argues that the development process is essentially instruction-oriented and can be represented by software programs. That is, software processes, in effect, consist of a number of defined sequential instructions, which can be automated through software programs. There is however an argument that the development process is too complex and a too human-oriented activity to be treated and represented as software programs.

The argument about the concept of development processes, and the suitability of their characteristics to be represented in software programs, was proposed in an influential paper (software processes are software too) by Osterweil [1987]. In this, and a subsequent paper [Osterweil 1997], he argued for rigorous process descriptions to guide the key software processes, in which programming techniques and formalisms can best facilitate the task. The proposal was for software processes descriptions to be expressed and formed using programming in an activity referred to as 'process programming' or 'process modelling'. As a proof of concept he helped develop a process programming language, called Little-JIL [Cass et al. 2000], which is a graphical method for process programming and defines processes that coordinate the activities of autonomous agents and their use of resources during the performance of a task. An example of instruction-oriented software processes is illustrated in a process function shown in Table 2-17.

```
Function All_Fn_Perf_OK(executable, tests):
  declare executable executable_code.
  tests testset,
  case, numcases integer,
  result derived_result:
  All_Fn_Perf_OK := True;
  For case := 1 to numcases
    (Comment. execution of the testcases in a testset array)
    derive (executable, tests[case].input_data, result)
    (Comment. Compare results with expected behaviour; abort if any test execution does meet expectations)
    if Not resultOK (result, testcase[case].req_output)
    then All_Fn_Perf_OK := False;
    exit;
  end loop;
end All_Fn_Perf_OK;
```

Table 2-17 Example of a process function (program) Osterweil [1987]

The program demonstrates how the testing process might be automated through a function, which loops through a set of instructions until all *test cases* are executed and the results are recorded. Materialisation of process in this sense might further benefit software measurement through the generation of new metrics (e.g. product size as no. of objects in process program). Perhaps the most important benefit of such process programming is that it would offer the possibility of reusing software processes.

There is however evidence that process programming is inappropriate and inapplicable for large applications (programming in the large) [Lehman 1987]. The very existence of a programming language places constraints on how a problem may be solved and limits human creativity. Furthermore, every stage and step of the programming process requires thought, analysis and review of the earlier steps, which could mean repeated refinements, or even redoing of the earlier models and steps, as the understanding of what came before evolves - this would be impossible in a procedurally formatted process program [ibid]. Process programs may be feasible for small, well-structured, well-defined and understood applications (i.e. compilers). However, in cases where the software project is large and complex, where not all parameters could be well defined, using process programs for the complete development process is neither feasible nor practical.

Since the publication of Osterweil's paper [1987], software process modelling has gained much interest in the software community among both academic researchers and practitioners [Madachy 2008]. However, while some

software process activities can be expressed in 'well-defined' and sequential textual instruction that can be automatically and systematically executed in software programmes (as illustrated in Table 2-17), this would not be currently appropriate or possible for complex and human-oriented software processes. While representation of development process in textual instructions, upon which a software program can act, is greatly useful in systematically organising and executing software development processes, complex processes cannot often be properly represented in this way. Such complex processes often require human judgements and decisions to determine the required process tasks in real-time, and cannot be completely pre-planned and programmed.

In the above discussions, an association between patterns and tasks was established. This would imply that patterns could benefit from task analysis. This is discussed in the next section.

### 2.6.5 Can Patterns Benefit from Task Analysis?

As was demonstrated above, patterns contain tasks and can be themselves considered tasks, when they are used in a sequence. The way a number of patterns are selected from a pattern language, to form a pattern sequence to solve a particular problem, may be governed by the method used for their selection. This may be best accomplished using task analysis in general and hierarchical task analysis in particular, due to the hierarchical nature of the patterns.

A number of software and non-software patterns were discussed above in relation to the hierarchical and task-oriented nature of their solution elements. Such decomposition of a higher task into smaller lower level tasks is an important element of Hierarchical Task Analysis (HTA), which is discussed in the following section (Section 2.6.6). Furthermore, the method in which the solution in a pattern is decomposed into a number of tasks may also be explained and analysed by task decomposition in HTA.

From one perspective, there appears to be a relationship between task analysis and patterns that has not been yet explored in the literature. That is, if a pattern is defined in terms of a task that is to be performed to achieve a goal, then task analysis becomes relevant and useful. There are three ways in which hierarchical task analysis can be applicable in making a pattern or a pattern language optimal and improve the quality of the solution it provides. These are:

1. *Developing individual patterns:* The tasks and the order and sequence in which they should be performed in a pattern, whose solution element includes a number of tasks.
2. *Developing a Pattern Language:* Clustering and grouping a set of related tasks into many individual interlinked patterns.
3. *Developing a pattern sequence (i.e. a construct or language):* The patterns selected from a pattern language, to solve a problem, and the order and sequence in which they should be applied.

Task analysis may be most useful in circumstances where a problem is to be solved using patterns from one or more pattern languages consisting of a multitude of single patterns. In such circumstances, a proper analysis of the problem to be solved and the goals and objectives to be attained is crucially important in choosing the right patterns to apply in the right combination and order.

The hierarchical nature of patterns, as demonstrated in a number of examples above, helps the pattern solutions to be of higher quality in terms of clarity and structure, which can be explained by a method of task analysis called 'Hierarchical Task Analysis' (HTA). In the following section, the HTA method is discussed.

### 2.6.6 Hierarchical Task Analysis

Annett and Duncan [1967] proposed the Hierarchical Task Analysis to evaluate organisations' training needs by decomposing complex training tasks into a set of task components, which could then be trained. In this proposal, a task is broken down and sequenced from top to bottom, thereby showing a hierarchical relationship amongst the tasks. HTA's underlying technique is hierarchical decomposition, which analyses and presents the behavioural aspects of complex tasks. It decomposes the tasks into subtasks, operations or actions and a structure chart is then used to represent the tasks graphically. HTA concerns identifying and categorising tasks and involves the notion of goal and task. A goal is simply defined as something to be achieved. Attainment of a goal requires the completion of a plan that involves a number of individual tasks. A task is an activity that must be carried out to achieve a goal. The purpose of the HTA is to decompose the higher level tasks into lower level tasks (i.e. sub-tasks), each of which will satisfy a sub-goal. Carroll [2000] states that a task, has to be real and not

divorced from actual user practice, be central to multiple user activity so that addressing it may have general benefit, and require near-perfect execution.

HTA has been shown to be capable of providing useful descriptions of a variety of tasks in many contexts. It has been shown to aid human decision-making in the design of teams and jobs, operating procedures, selection methods, interface design, training, and reliability assessment [Ormerod and Shepherd 2004]. In interface design, HTA provides a model for task execution, which enables designers to envisage the goals, tasks, subtasks, operations, and plans for users' activities. It is based on functional, rather than behavioural or psychometric constructs, and uses a fundamental unit called an *operation*. The key features of an operation are the conditions under which the goal is activated and satisfied, and the actions, which need to be performed to attain the goal [Diaper and Stanton 2004]. These actions may themselves be defined in terms of sub-goals. For example, thirst may be the condition that activates the goal of having a cup of tea, and sub-goals could include obtaining boiling water, a teapot, a tea bag, a cup and so on.

At the highest level, a task consists of an operation which is defined in terms of its goals and which is measured in real terms of production units, quality, or other criteria [Annett 1971]. The operations can be broken down into sub-operations in a hierarchical relationship, each defined by a sub-goal. Therefore, to satisfy a goal in a hierarchy, its immediate sub-goals have to be satisfied, and so on. The rules that govern the relationship between the immediate super-ordinate and its sub-ordinates guide the sequence with which each sub-goal is attained [Stanton 2006]. These rules are facilitated by a number of notations, as depicted in Table 2-18.

Symbol	Meaning	Example	Description
>	Then	1>2>3>	1 Then 2 then 3
+ &	And	1+2+3	1 And 2 And 3
/	Or	1/2/3	1 Or 2 Or 3
:	Any of	1:2:3	Choose any
K ? >	If condition	K? Y>1 N>3	If K Then 1 Else 3

Table 2-18 HTA notations

There are a number of proposed guidelines in the literature for conducting HTA, which are fundamentally similar [Annett 2004]. The basic heuristics for conducting HTA is as follows [Hone and Stanton 2004]:

1. Define the purpose of the analysis
2. Define the boundaries of the system
3. Assess a variety of sources of information about the system to be analysed
4. Describe the system goals and sub-goals
5. Try to keep the number of immediate sub-goals under any super-ordinate goal to a small number (i.e. between 3, and 10)
6. Link goals to sub-goals and describe the condition under which sub-goals are triggered
7. Stop re-describing the sub-goals when you judge the analysis is fit for purpose
8. Try to verify the analysis with a subject matter expert
9. Be prepared to revise the analysis

For example, in instruction design, the instructional designer breaks down a task from top to bottom to show a hierarchical relationship amongst the tasks, and then instruction is sequenced bottom up. A task at a higher level cannot be performed until the subordinate tasks are all carried out. Once, as a result of the decomposition, a comprehensive list of the tasks that make up a job or function are available, three major steps need be performed to construct a hierarchy. These are: 1) Group and cluster the task that bear close resemblance to each other, 2) Organise tasks within each group to show the hierarchical relationships, and 3) Consult with a subject matter expert to determine the hierarchy's accuracy.

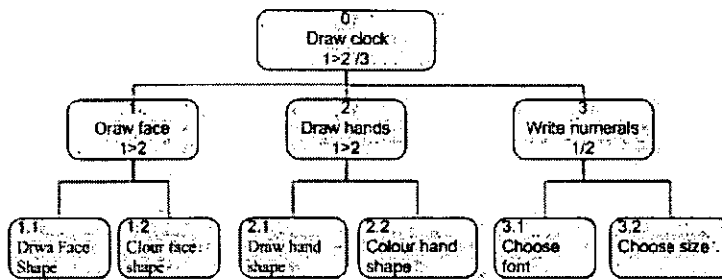


Figure 2-5 HTA for drawing a clock

Examples of HTA, in drawing a clock and inspecting instruments in an acid distillation plan, is illustrated in Figure 2-5 and Figure 2-6 respectively. Each task has a unique number and a plan, which states the format under which the sub-tasks are to be executed. For example, the *draw face* task (1) is dependent on tasks (1.1) and (1.2) both of which need to be carried out in order to achieve task (1).

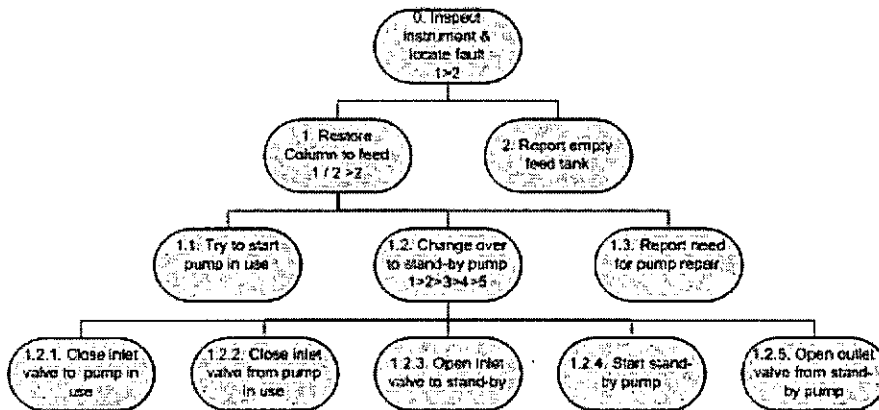


Figure 2-6 Section of the goal hierarchy for an acid distillation plant operator's task [Annett 2004]

Having briefly discussed and outlined the principles of the HTA, the following section discusses ways in which HTA can be utilised in patterns.

### 2.6.7 Application of HTA in Patterns

As illustrated earlier in Table 2-11, a pattern sequence consists of a number of individual patterns strung together to solve a problem. HTA can offer the designer the ability to structure the tasks in the pattern sequence more specifically and systematically. Using HTA it is possible to develop a pattern sequence, where the patterns involved can be set to be performed under certain conditions or in a certain order. This would make the developed pattern sequence (solution) more specific to a defined set of problems or group of problems.

The tasks involved in the *implementation (construction) phase* of a development lifecycle can be decomposed using HTA, as illustrated in Figure 2-7. Each of these tasks (operations) could be defined by a process pattern. In this example, the modelling is achieved by satisfying the subordinate sub-goals.

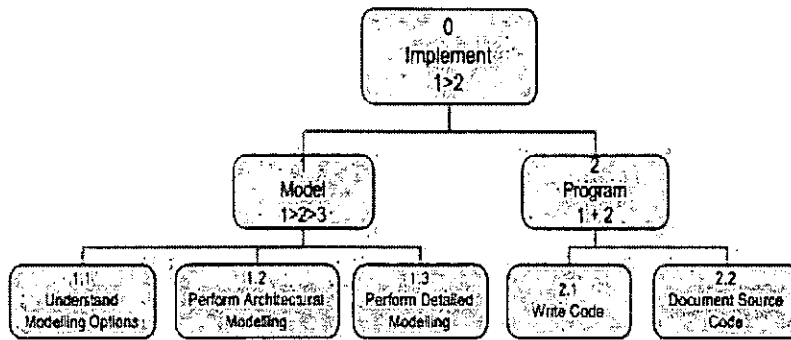


Figure 2-7 An example of task hierarchy for the Implementation phase where each task could be performed by a process pattern

The HTA chart in Figure 2-7 depicts activities required in the implementation phase of a development process. Here the HTA chart is a representation of a process pattern, which involves the engagement of other patterns to achieve the goal of performing the tasks in the *Implementation* phase of a software development lifecycle. This could be presented to a pattern language as a pre-defined pattern. It would however be possible to create a new *Implementation* patterns with modified process tasks, based on the pre-defined pattern. This would give the designer numerous options to construct solutions that are specific to a problem. For example, the *implementation* pattern, depicted in the HTA chart Figure 2-7, presents a solution in terms of the tasks required to be performed in the Implementation phase, which can be used as it is defined. However, one might need a solution that involves other process activities that are not covered in this definition (e.g. optimising code) in the Implementation phase. To do this one would create a new implementation process pattern based on the one predefined, and add the extra process activity. The new pattern can then be added to the pattern language.

However, in addition to the option of creating new patterns, a solution can be devised, by linking one or more pre-defined patterns selected from a pattern language to create a *pattern sequence*. The task analysis methods can be used to determine the sequence of patterns needed to solve the problem.

		Process Pattern Hierarchy Levels				
		1	2	3	...	n
Process Patterns	1	P <sub>1,1</sub>	P <sub>1,2</sub>	P <sub>1,3</sub>	...	P <sub>1,n</sub>
	2	P <sub>2,1</sub>	P <sub>2,2</sub>	P <sub>2,3</sub>	...	P <sub>2,n</sub>
	3	P <sub>3,1</sub>	P <sub>3,2</sub>	P <sub>3,3</sub>	...	P <sub>3,n</sub>
	m	P <sub>m,1</sub>	P <sub>m,2</sub>	P <sub>m,3</sub>	...	P <sub>m,n</sub>

Table 2-19 Process Pattern language

Task analysis can be employed to generate pattern sequences that can use individual patterns from all the hierarchy levels of a pattern language. Theoretically, there could be *n* levels of hierarchy of process pattern types. For each hierarchy level there could be *m* number of single patterns; the lower the hierarchy level of the pattern type, the higher the number of single patterns it would contain. This is depicted in Table 2-19 where P<sub>m,n</sub> denotes pattern *m* of the hierarchy level *n*. However, in practice there are only a few hierarchy levels, depending on the granularity of the defined pattern types. For example, Ambler [1988] defines three levels of hierarchy (Phase, Stage, and Task pattern) where there are only a few patterns at the highest level (i.e. development phases) and tens of patterns at the third highest level (i.e. tasks).

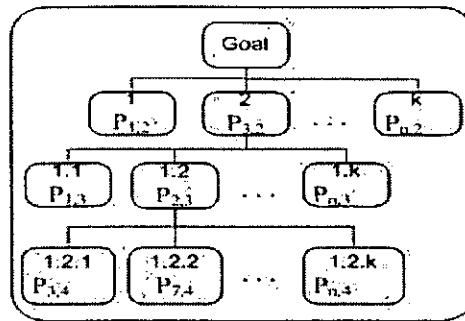


Figure 2-8 Example of a pattern sequence

An example of a possible pattern sequence, to produce a solution to a specific problem using patterns in a pattern language, is given in an HTA chart format in Figure 2-8. It demonstrates how patterns of different hierarchical levels are linked using HTA to form a pattern sequence to solve a specific problem. The following are examples of a number of possible pattern sequences, each designed to solve a specific problem. Each pattern sequence contains a number of single patterns that are to be executed sequentially.

Seq\_1 = (P<sub>1,2</sub>) (P<sub>5,2</sub>) (P<sub>9,3</sub>) (P<sub>1,4</sub>) (P<sub>5,4</sub>)

Seq\_2 = (P<sub>1,2</sub>) (P<sub>5,2</sub>) (P<sub>1,2</sub>)

Seq\_3 = (P<sub>1,2</sub>) (P<sub>5,2</sub>) (P<sub>9,3</sub>) (P<sub>1,4</sub>) (P<sub>12,4</sub>)

The pattern sequence Seq\_1 above involves the application of the patterns (P<sub>1,2</sub>), (P<sub>5,2</sub>), (P<sub>9,3</sub>), (P<sub>1,4</sub>), and (P<sub>5,4</sub>) in the stated order. A powerful utility of HTA that can be leveraged in developing pattern sequences is the ability to include conditional statements. That enables the creation of a dynamic pattern sequences where the sequence of patterns to be executed is not pre-defined. The series of patterns to be executed within the pattern sequence would be dependent on some conditions. A specific example is presented in Figure 2-9, where the order in which patterns are to be executed is variable. For example, to accomplish the goal, pattern P<sub>1,2</sub> has to be applied followed by pattern P<sub>5,2</sub>. However, to accomplish pattern P<sub>5,2</sub>, either P<sub>6,3</sub> or P<sub>9,3</sub> can be applied.

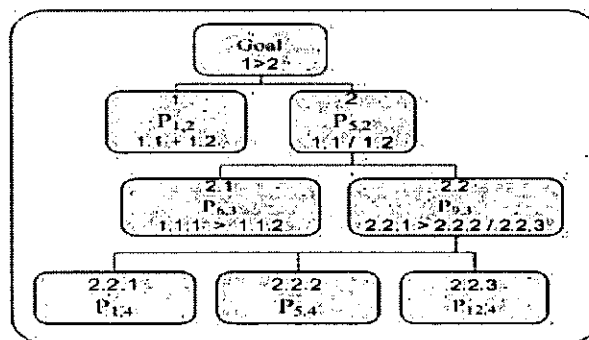


Figure 2-9 An example of a pattern construct using HTA

As the repository of software patterns grows and pattern languages could contain hundreds of patterns, a systematic method of selecting and implementing appropriate patterns, is necessary. Furthermore, often individual patterns are unable to provide a complete solution to non-trivial problems where the application of a sequence of patterns would be necessary. Some methods, such as using grammar in systematic selection of patterns, have been proposed in the literature [Zdun 2007]. In this section, it has been demonstrated how the methods of Hierarchical Task Analysis can be employed in constructing pattern-based solutions through pattern sequences, based on a methodical selection and combination of patterns. The important advantage of this system is the provision of conditional predicates (as demonstrated in Figure 2-9) which enables the construction of more detailed and specific solutions. Furthermore, the systematic and well-defined pattern sequence generation based on the proposed HTA methods, would make it suitable and feasible to create tools to facilitate pattern sequence creation.

### 2.6.8 Process Patterns Employed in the Experimentation

In this study, an experimental research was conducted to investigate the effectiveness of process patterns. The experiment utilised edited versions of a number of process patterns, which came from a variety of sources, including [Ambler 1998, 1999], [D'souza and Wills 1999], [Storrie 2000], [Coplien 1995]. The way in which the process patterns were made available to the experiment subjects is discussed in Section 5.5.3, and Section 5.75.7. The 'Appendix B. Patterns' presents a sample of the patterns used for this experimental study.

## 2.7 Summary

In this chapter the pattern concept, as applied in software engineering, was discussed. As designing and constructing architectural work has many similarities to software design and construction, the pattern concept, which was originally conceived for architectural design, has proven to be applicable and useful in software development. There are various definitions for patterns, but the simplest and widely used one is that 'A pattern is a proven solution to a problem in a context'; although one can argue that this simplified definition is not comprehensive enough (e.g. lacks recurrence aspect) to properly define pattern.

There are many topical issues in patterns currently being discussed within the pattern community. These include issues such as 'whether patterns should be formalised', or whether software patterns comply with the principles of the pattern concept. These issues were discussed in detail.

The concept of patterns has been applied to software engineering in various fields. They have been applied to both product and process aspects of software development. Software patterns, where the emphasis is on architectural and code level structure of the software application, are referred to as *design patterns*. Software patterns, which define and describe the process involved in developing a software application, are referred to as *process patterns*. In design patterns the works of Gamma et al. [1995] (Known as GoF), in which they captured and presented 23 design patterns, has been well received by the software engineering community. In the area of software processes, Coplien [1995] and Ambler [1998, 1999, 2002, 2005] have produced many established process patterns. Some of these process patterns have been used for the experimental research in this study. There have however been few studies investigating the utility and effect of patterns. This project addresses this issue by conducting an investigation of the utility and effect of patterns in software engineering.

The relationship between task analysis and patterns was discussed and, the possibility of using Hierarchical Task Analysis in pattern usage and pattern development was explored. It was shown that the hierarchical structure is one of the main aspects of the pattern concept. It has been further shown, through a discussion of the Hierarchical Task Analysis, that the presentation and analysis of tasks in a hierarchical manner proves advantageous in achieving the desired goals. Application of HTA, in the development of patterns and pattern sequences was explored and the benefits were outlined.

In the next chapter, the detail of a preliminary study in the form of two surveys, to evaluate the usage levels of patterns within both the architecture and software communities will be discussed.



---

## Chapter 3 Pattern Usage Surveys

---

### 3.1 Introduction

Survey research method is one way of obtaining valid scientific knowledge. The survey research provides a way of observing some phenomenon and forming, testing, and validating theories based on the observations made [Babbie 2001]. In this study, two preliminary surveys were utilised in order to understand architectural and software pattern issues and gauge the usage levels, and to help devising the research question. The objective, in both surveys, was to capture data on a number of constructs (variables to measure) through devising a number of questions in survey instruments. The data was to be provided by a sample of the population of interest (i.e. software development organisations, and architects), drawn through devised sampling methods.

The first survey, which is discussed in the first section of this chapter, aimed to determine issues concerned in architectural patterns, as well as their level of support within the architecture community, in an effort to make sense of the concept in the original environment. The survey investigates the views and opinions of the architects in the architecture departments in UK universities on the pattern concept. The second survey attempted to gauge the effect and value of software patterns in software development companies. In this survey, a number of software development companies were investigated to determine their use and application of patterns within their software development practices. This will be discussed in the second section of this chapter.

### 3.2 Architectural Patterns Survey

In this section, the survey on the use of architectural patterns, by architects in academia, is discussed. The survey's aim was to investigate the views and opinions of the architects within the architecture departments of UK universities on the pattern concept, and determine their popularity and usage in terms of the extent to which they are taught in universities. The aim was to determine the difficulties and pitfalls that have damaged the prospects of architectural patterns, and to discuss whether such issues and difficulties would apply, and could prove damaging, to software patterns.

#### 3.2.1 Motivation

The concept of pattern languages in architecture has engendered much controversy within the architecture community. While some universities taught the subject, many others completely avoided it. Since the pattern concept was conceived in the field of architecture, and has therefore a longer history in this field, a study of how patterns are perceived and utilised in architecture may indicate what could happen to software patterns and provides valuable lessons. There appears to be no published surveys on the usability levels of pattern languages by architects.

The objectives in this survey were to determine the popularity of architectural patterns, within the architecture community (in academia), and determine their views and opinions on the strengths and weaknesses of the pattern concept. The overall aim was to determine whether software patterns are likely to be influenced or undermined by the same or similar difficulties found in the architectural patterns.

#### 3.2.2 Survey Details

All UK universities, with an architecture department, were invited to participate in this survey. At the time of this survey, there were found to be 36 such departments running undergraduate and/or postgraduate courses. In this survey, emails and follow-up telephone calls were used to contact the samples, sending each a questionnaire (in the form of a letter) to complete and return. The questionnaire was emailed to the heads of the architecture departments in the sample universities. They were invited to fill-in the questionnaire themselves, or pass it on to another architect in their department. The questionnaire contained only two questions, to make it as inviting as possible for participants to reply. Here are the two questions:



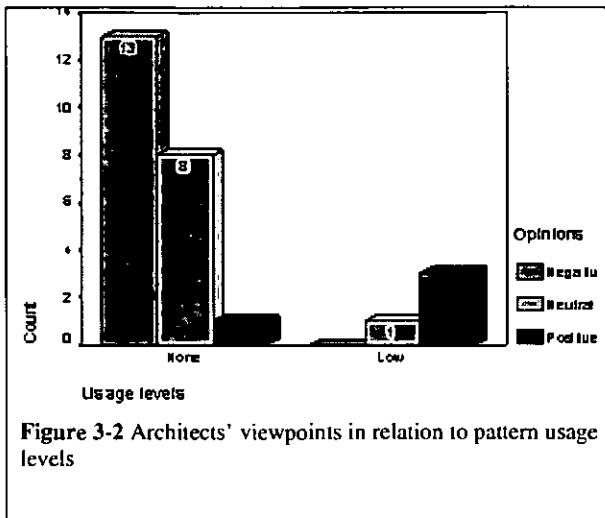


Figure 3-2 Architects' viewpoints in relation to pattern usage levels

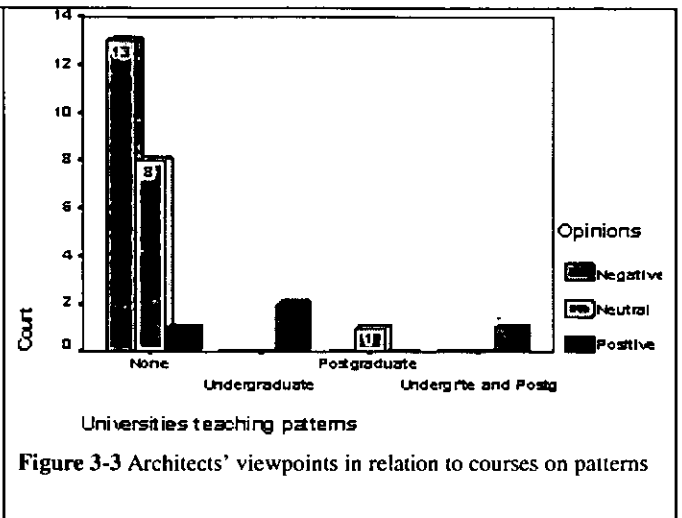


Figure 3-3 Architects' viewpoints in relation to courses on patterns

There appeared to be a relationship between *pattern usage* and architect's *viewpoints*. A correlation analysis was carried out which is depicted in Table 3-1 and the scatter plot in Figure 3-4 (Correlation analysis is discussed in 7.5). There is a statistically significant positive correlation between *pattern usage* and architect's *viewpoints* with Corr. Coef.  $r=0.494$ , and Significance.  $P=0.02$ . Correlation is significant at the 0.05 level (2-tailed). Therefore, as pattern usage increases, architects' viewpoints also increase proportionally. Furthermore, an increase in architects' viewpoints will be reflected in a proportional increase in pattern usage.

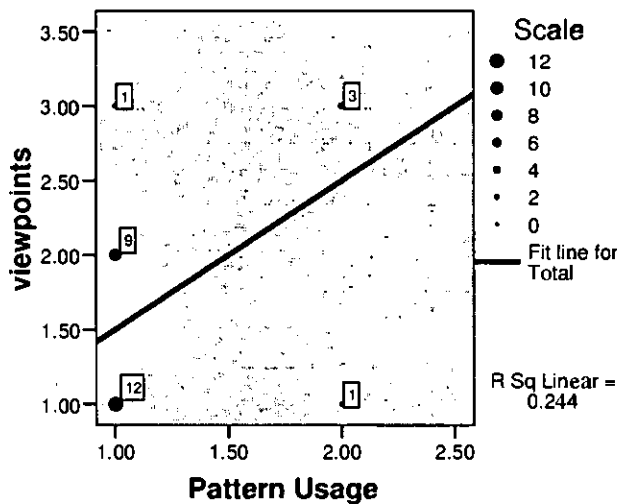


Figure 3-4 Correlation between pattern usage and architect viewpoints

		Pattern Usage	viewpoints
Pattern Usage	Pearson Correlation	1	.494
	Sig. (2-tailed)		.010
	N	26	26
Viewpoints	Pearson Correlation	.494	1
	Sig. (2-tailed)	.010	
	N	26	26

Table 3-1 Correlation between pattern usage and viewpoints

In the following section, the qualitative data collected in the survey is discussed.

### 3.2.3.2 Qualitative Results

Senior representatives of architecture departments of UK universities were asked to express their opinion about architectural patterns in this survey. While many Heads of departments kindly responded to the survey themselves, some dedicated the work to someone in their department. Some of the views expressed are listed in the 'Appendix D. Results'.

It is clear from the comments that the overwhelming majority of commentators did not value architectural patterns as a contemporary, forward-looking concept, from which new ideas and works could be generated. It appears that the architects believed that Alexander's concept of pattern languages were rather old fashioned and opinionated. It is interesting that there seemed to be much opposition to Alexander, and his concept of pattern language, within the architecture community. This can be clearly felt by quotes such as "I co-ordinate first year studio, and steer well clear of Pattern Language". Some, however, believed that the pattern languages would become popular in the future. These results are further discussed in the next section.

### 3.2.3.3 A Discussion of the Results

The results presented above indicate that there were a number of issues with architectural patterns, which concern and dissuade architects to actively incorporate patterns in their design practice. It is interesting that the research provides evidence that the pattern concept does not have much support in the architecture community for which it was conceived. The results indicate that the general views of the architects, within the academia, seem to be that Christopher Alexander's philosophy of pattern languages are "rather old and tired ideas" which stifle creativity in architecture. Given the concerns and objections architects express about architectural patterns, should the software community be concerned that the pattern issues and pitfalls, raised by the architects, could, at some point, catch up with software engineering and render software patterns effectively harmful? The majority of the surveyed architects criticised patterns as being anti-creativity, authoritarian, unscientific, and old fashioned. The question is whether the problems and issues that caused architectural patterns to be unpopular within the architecture community, could also prove damaging to the prospects of software patterns in the future. Based on these results, this section will discuss whether the issues raised in architectural patterns could also apply to software patterns, now or in the future.

#### Anti-creativity

A large proportion of the respondents in the survey expressed the view that architectural patterns are anti-creativity and authoritarian. This reflects the general views of many architects within the architecture community [Saunders 2002] [Kohn 2002] [Eakin 2003]. The argument is that patterns seem overwhelmingly authoritarian telling the reader what must be done in a controlled manner. This is in conflict with individual freedom and maximum choice and, therefore, patterns stifle creativity. It is furthermore argued that patterns are prescriptive and require architects to design according to some specific set of rules demanded in the pattern. Such subservient adoption of patterns, critiques argue, would effectively encourage architects to copy designs rather than to try generating creative designs.

However, in reality, architectural patterns do not necessarily restrict creativity, nor do they hinder artistic freedom [Salingaros 2000]. It all depends on how the pattern is used and employed in design. Patterns aim to bring to the attention of the designers the designs that have been proven to work; solutions that are timeless. Furthermore, there are an indefinite number of ways that the architectural patterns can be put together to generate new designs, giving the designer the choice and freedom to express their creativity. In effect, by imposing constraints, the patterns eliminate a large number of inferior possibilities, while allowing an infinite number of possible plausible designs. Therefore, accusations and criticism of architectural patterns as being a hindrance to self-expression, which has caused architects to resist using patterns, seems to be unjustified. It should also be borne in mind that while freedom of expression is important, the primary function of architecture should be to provide structures that are comfortable and useful. In the current architectural paradigm, however, it seems the emotional and physical comforts of the user are of only minor importance [Salingaros 1999]. Patterns help architects to create designs that are useful as well as offering some levels of freedom of expression.

The creativity accusation is, however, far less significant in software patterns. While in architecture human creativity in presenting artistic structures is important, in software engineering the emphasis is not so much on the artistic aesthetics of the software. What are crucial in software development are designs that are robust,

efficient and have been shown to work in practice. Furthermore, while software patterns provide the overall and a high-level solution to a problem, the exact implementation of the software pattern is not defined by the patterns. This gives the software architects and developers the ability to be creative in implementing the pattern solution.

Based on the importance attached to creativity (as expressed by the respondents in the survey), we recommend that software pattern authors should endeavour to generate patterns that would offer pattern users maximum choice in implementation styles, while not undermining the quality, clarity, and un-ambiguity of the patterns.

### **Outdated and old fashion**

A further criticism of the architectural patterns, expressed by the survey respondents, was that the architectural patterns were old fashioned and out of date. The criticism is that architectural patterns are set in, and reflect, the past and therefore, while they would be applicable and suitable for their time, they are out of place for the modern era. For example, patterns that architects used to build cathedrals and other historic buildings a few centuries ago are not necessarily suitable and desired by the today's modern society. Times have changed and architecture has and will continue to change.

While architecture is a discipline that is thousands of years old, software engineering is relatively young. Therefore, while architectural patterns could reflect structures that are hundreds or even thousands of years old, software patterns reflect designs and solutions that are merely a few decades old. For example, pattern in Object-Oriented programming which is currently the most popular programming paradigm have an age of about two decades. This is, however, not to say that software patterns do not age or outdate. Software patterns over time could be obsolete, due to many reasons such as the technology on which they are based. As the technologies change and improve, patterns based on the older technologies will outdate and die out. For example, software patterns on the Waterfall Process Model [Royce 1970] that would have been perfectly valid to be used a couple of decades ago (partly due to the insufficient computing resources), would now be almost obsolete in their original form for many types of software development projects, as the technologies and computing resources have changed and improved.

There is, therefore, nothing wrong in software patterns getting old and outdated. It is inevitable that many software patterns, especially domain and platform specific patterns i.e. J2EE patterns, will have an expiry date in terms of the validity of the solution they provide. We recommend that pattern authors should ensure that they fully state the scope and the context in which the patterns they are producing are applicable. This would ensure that the pattern user would know if the patterns they were going to use would work for the specific problem to which they are applied. Such details, in scope and context, would further inform the pattern users whether the pattern would work, if some underlying technologies changed (e.g. whether they are platform dependent). Therefore, technically, no software pattern (new and old) should run the risk of being misapplied, if the context under which the pattern is applicable (i.e. pattern's *context* element) is properly and fully defined.

### **Unscientific**

A further criticism of the architectural patterns, expressed by the architects in the survey, is that there is little proof for the theories and assertions and that the main evidence has been the author's own work, opinion, and imagination. The argument is that the architectural patterns are opinion-based, subjective, and not scientifically validated. Some [Saunders 1999, 2002] have argued that the architectural patterns are based on observation, without methodology. Critics, further point to some of the assertions in the architectural patterns such as "we guess ..." or "Several studies show ..." and suggests that such statements has little scientific support [ibid].

In software patterns, however, the scientific extraction and validation process may differ, according to the type of software pattern in question. Technical software patterns may be more scientific, due to their technical nature. While observation is the main method by which such patterns are extracted and formed, they are easier to verify and validate. The validity of patterns can be judged by the applications that implement them. If, for example, a software pattern has been implemented in three applications that have all operated successfully for sometime, then the pattern can be considered validated. In fact the software pattern community has recommended, what is called, "rule of three" which requires each pattern to have been observed to operate successfully in three different situations. As well as being observed in a number of successful applications, software patterns can be further validated by testing and evaluating them for various quality attributes in specifically written applications. For example, in addition to noting the applications that had successfully implemented the patterns, Gamma et al. [1995] further validated their patterns through a specifically written software application.

While such validation methods are possible for software patterns dealing with software design (i.e. design patterns), process and human-oriented software patterns, such as process/organisation patterns, are more difficult to validate scientifically. Often the only validation offered in these types of software patterns is the pattern author's experience [Coplien 1995] [Ambler 1998]. Experimentation, as a validation method for process and organisational patterns, has been proposed in this research project [Estabraghy and Dalcher 2007a]. Therefore, process and human-based patterns can also be scientifically validated (albeit more time consuming and expensive than design patterns). Validity of process patterns can also be checked by their implementation in successful software development projects.

Given the criticism of patterns, as being unscientific and invalidated, expressed in the survey, we recommend that software pattern authors should do more to ensure that the software patterns they author are scientifically extracted and are fully validated. A strict adherence to the "rule of three", as well as evidence of independent validation tests (where possible), are recommended.

Having discussed the survey on the architectural patterns in this section, in the following section the second survey to investigate the effect and utility of software patterns is discussed.

### 3.3 Survey of Software Organisations

In this section, the survey research method on the use and application of patterns in software development projects in industry is discussed.

#### 3.3.1 Motivations

Over the years, the application of the pattern concept in software engineering has been widely written about and investigated through books, journals and conference literature. The purpose of this investigation was to carry out a study of software patterns to evaluate their impact on software development practices in software development organisations. While academics develop theories leading to the introduction of new technologies, it is the software industry that implements the theories in practice. Studies show that over-dependence on unreliable new technology is one of the main causes of software project failure [Glass 1998]. It is therefore important to gauge the software industry's viewpoints, experiences, and reactions on any new and introduced piece of technology (such as software patterns) with the aim of evaluating their utility and improve them accordingly. In this study, a survey research method was designed and implemented in order to investigate the experiences and opinions of software development organisations on the impact and application of patterns in their development practice.

#### 3.3.2 Related Work

There have been numerous publications in the form of books, journals and conference papers on software patterns (see for example, [Buschmann et al. 1996, 2007] [Coplien et al. 2005] [Gamma et al. 1995] [Fowler 1997, 2002]). There are many claims, with some empirical verification, that software patterns can capture the essential component of a design, be used to record and reuse best practices, and provide the vocabulary for communicating complex concepts effectively [Gamma et al. 1995][Beck et al. 1996][Buschmann 2007]. While there have also been some experimental studies to investigate the impact of software patterns on software development projects [Prechelt 2001, 2002] [Unger and Tichy 2000], there is very little published research investigating the impact and value of software patterns in the software industry. A recent survey research, conducted at the IBM [Manolescu et al. 2007] (see Section 2.2.8), which surveyed over 70 software development organisations, indicates a widening gap between pattern experts and the average software developer and designer. The study found that while software patterns written by experts had included many types and aspects of software development (e.g. process, architecture, and integration), software developers and practitioners had only concentrated their efforts on the 23 design patterns introduced by Gamma et al. [1995]. Many of the findings in this survey research correspond to the works of Manolescu et al. [2007], which strengthens the results and conclusions reached by both studies. The details will be further discussed in this chapter.

The Patterns\_Central [2005] website ran a survey on software patterns. The survey showed that 50% of the respondents believed that patterns were useful. The survey also indicated that 59% of respondents used patterns. However, 32% of the respondents believed that patterns were either misused or misunderstood. For further details of the survey's results, see Appendix D. Results.

### 3.3.3 Samples and Sampling Method

Sampling is a process of selecting the samples for the survey. A number of sampling techniques were considered for their suitability for this research and the 'Systematic Random Sampling' method was chosen. Systematic Random Sampling is appropriate when the selection of a sample needs to be taken from a list [Sapsford 2007] (i.e. a list of software development organisations). A sampling fraction (k) was calculated by dividing the population (i.e. software development organisations) by the required sample size of 500. A random number was then selected between one and k, and beginning with the selected random number every k<sup>th</sup> unit in the list was selected as a sample member. Therefore, 500 organisations were selected to participate in this survey, 67 of which accepted.

The sample for the survey was selected from a complete list of the software development organisations in the U.K that are listed in the Kompass Business Directory [Kompass]. Although most large organisations have a computer department, and may develop software for their internal use, this survey aimed to specifically study those organisation that develop software for sale and are listed as software development organisations in the directory. Based on the software development organisations listed in the Kompass directory, a sample of 500 software development organisations were randomly selected, using the sampling method discussed above, to be surveyed. The sampling unit (or unit of analysis) of the study is an individual software development organisation. A member of the development team, from participating organisations, provided the data for the survey as the representative of each sampled organisation.

A number of methods of collecting data [Babbie 1990] were employed in this survey. These included one-to-one interviews with representatives of the 67 sample organisations. Where such interviews were not achievable or available, telephone interviews were conducted to collect the required data (this was the most used method). A questionnaire was also hosted on a specific website through which the participating organisations could complete and submit their responses. The collected data, captured through the stated methods, were recorded in a database, which was subsequently used by the SPSS statistical package for analysis.

### 3.3.4 Survey Instrument

The survey instrument was designed to capture the data of interest (i.e. constructs) through a number of specific and unambiguous questions set in a questionnaire. The data of interest were of the following major type:

**Organisation's type and attributes:** These types of questions capture data about the characteristics of the participating organisations (e.g. number of employees, ISO 9000 registered)

**Pattern usage:** Questions about the practice of using and implementing patterns (e.g. type of patterns used, effect of patterns on reliability, efficiency etc.)

**Pattern development:** These types of question aim to capture data from organisations that develop patterns (e.g. type of patterns developed and whether developed patterns are published externally)

**Non-Pattern usage:** This type of question captures the responses of organisations that do not use patterns (e.g. reason for not using patterns, any plans for employing patterns).

The questions were presented to the participant organisations through the data collection methods discussed in 'data collection methods above'. The Table 3-2 presents an example of the questions that appear in the survey instrument. The complete survey instrument is in 'Appendix E. Survey Questionnaires'.

What do you believe to be the effect of application of patterns on the following software quality attributes?								
<b>Reliability</b>	Positive	<input type="checkbox"/>	Negative	<input type="checkbox"/>	Neutral	<input type="checkbox"/>	Don't Know	<input type="checkbox"/>
<b>Usability</b>	Positive	<input type="checkbox"/>	Negative	<input type="checkbox"/>	Neutral	<input type="checkbox"/>	Don't Know	<input type="checkbox"/>
<b>Changeability</b>	Positive	<input type="checkbox"/>	Negative	<input type="checkbox"/>	Neutral	<input type="checkbox"/>	Don't Know	<input type="checkbox"/>
<b>Interoperability</b>	Positive	<input type="checkbox"/>	Negative	<input type="checkbox"/>	Neutral	<input type="checkbox"/>	Don't Know	<input type="checkbox"/>
<b>Efficiency</b>	Positive	<input type="checkbox"/>	Negative	<input type="checkbox"/>	Neutral	<input type="checkbox"/>	Don't Know	<input type="checkbox"/>
<b>Reusability</b>	Positive	<input type="checkbox"/>	Negative	<input type="checkbox"/>	Neutral	<input type="checkbox"/>	Don't Know	<input type="checkbox"/>
<b>Testability</b>	Positive	<input type="checkbox"/>	Negative	<input type="checkbox"/>	Neutral	<input type="checkbox"/>	Don't Know	<input type="checkbox"/>
<b>Portability</b>	Positive	<input type="checkbox"/>	Negative	<input type="checkbox"/>	Neutral	<input type="checkbox"/>	Don't Know	<input type="checkbox"/>
<b>Maintainability</b>	Positive	<input type="checkbox"/>	Negative	<input type="checkbox"/>	Neutral	<input type="checkbox"/>	Don't Know	<input type="checkbox"/>

Table 3-2 An example of the survey questions

### 3.3.5 Software Pattern Survey Results

A total number of 3751 companies were listed as software development companies in the Kompas business directory. Out of a sample size of 500, a total of 67 respondents were achieved. Given the organisations' low response rates at around 7% [Walonick 1997], this is an acceptable size and is comparable to other software engineering surveys (for example [Manolescu et al. 2007] [Tang et al. 2006] [Lethbridge 2000]). In the following sections, the details of the results of the survey instrument are presented and discussed. Some of the results are presented in 'Appendix D. Results'.

#### 3.3.5.1 Pattern Usage

As depicted in Figure 3-5, 40 out of the 67 (59.7%) respondents used patterns in their software development practices. The Figure 3-6 illustrates pattern usage in relation to organisation size. The figure shows that the size of the companies has an affect on the patterns usage level. Small companies with less than 10 staff formed the category with the least usage, with only 8.3% of the companies using patterns. One reason for this could be that, the smaller companies are more likely to be involved in the development of small and predominantly graphical-based web applications, where software patterns have currently minimal utility. Between 75 to 100% of the surveyed software development organisations over the size of 50 employees used patterns.

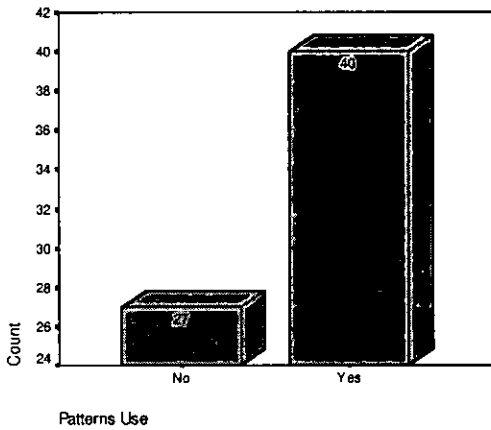


Figure 3-5 Companies using patterns

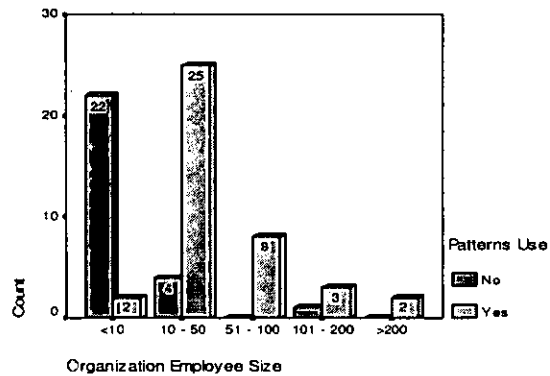


Figure 3-6 Pattern usage in relation to organisation size

The participants were asked to state their views on both *usability* and *usefulness* of patterns. The results are presented in Table 3-3 and Table 3-4. It is interesting that the respondents' views on both usability and usefulness appeared to correspond to their level of pattern *usage*. Therefore, the correlation between pattern usability and pattern usage variables was investigated which is shown in Figure 3-7 (Correlation analysis method is described in Section 7.5). The results show that there is a statistically significant positive correlation between the two variables (pattern *usability* and pattern *usage*) with Corr. Coef.  $r=0.65$ , and Significance  $P=0.000$ . There is also a statistically significant positive correlation between pattern *usefulness* and pattern *usage* with Corr. Coef.  $r=0.562$ , and Significance.  $P=0.001$ . The correlation is depicted in the scatter plot in Figure 3-8.

Pattern Usability (Ease-of-use)				
Pattern Type	Easy	Moderate	Difficult	Very Difficult
Analysis Patterns	0	1	0	0
Design Patterns	5	9	15	11
Process patterns	0	5	1	0

Table 3-3 Pattern usability results



**Correlation - Pattern Usability and Usage**

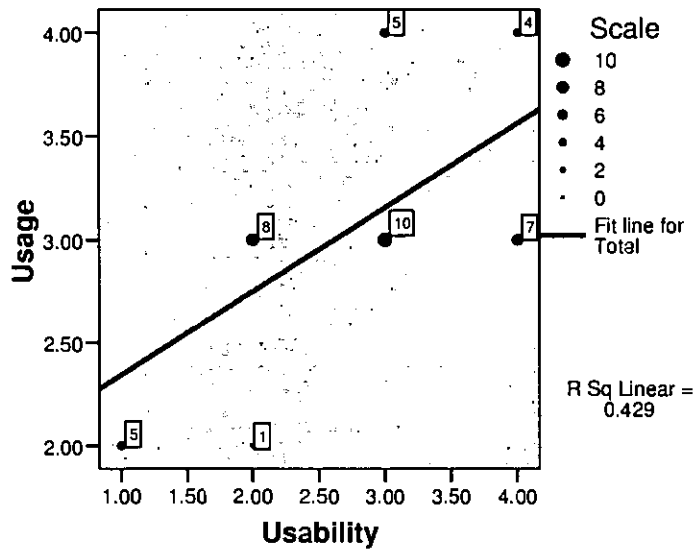


Figure 3-7 Correlation between pattern usability and pattern usage

Pattern Usefulness				
Pattern Type	Nil	Slight	Moderate	Considerable
Analysis Patterns	0	0	1	0
Design Patterns	2	2	15	21
Process patterns	0	2	3	1

Table 3-4 Pattern Usefulness Results

**Correlation - Pattern Usage and Usefulness**

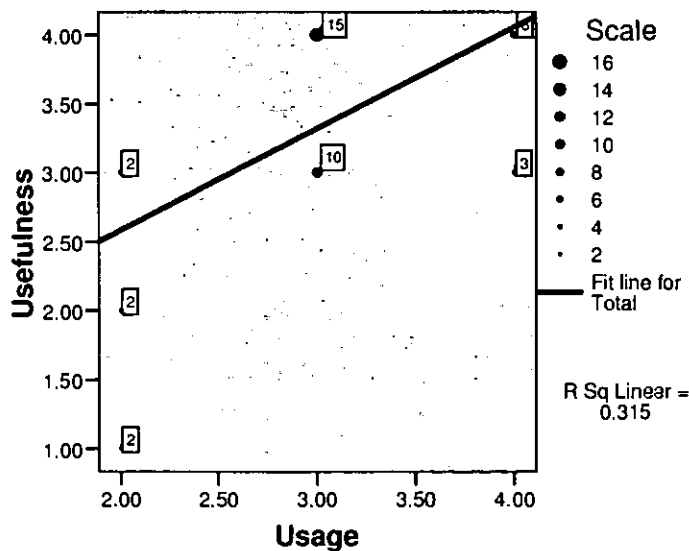


Figure 3-8 Correlation between pattern usefulness and pattern usage

The results indicate that the more the users find patterns easy to use, the more they use them. It appears that when practitioners have confidence and skills in using pattern, the pattern usage levels increases proportionally. It also appears that the more they use patterns, the more they are convinced of their usefulness. Therefore, there are some steps and actions that can be taken by the pattern community and software organisations in encouraging a wider use of patterns:

- **Provision of training:** One of the main reasons for not using patterns has been shown in this survey to be the lack of skilled practitioners. Manolescu et al. [2007] also found that only 10% of software developers in the surveyed organisations had been on a pattern-training course. If the level and quality of pattern usage is to increase substantially, provision of training for software engineers should be taken more seriously by the software organisations. We recommend that they should aim to schedule comprehensive pattern training programmes for their software engineers. Training would enhance the skills and confidence of software engineers to use patterns, which would proportionally increase pattern usage as shown in this study. Since, this study and others [Prechelt 2001, 2002][Beck 1996], have shown that pattern usage has a positive effect on software quality, any incurred training costs would be likely to prove valuable investments in terms of producing better quality software.
- **Provision of a wider choice of pattern:** A more comprehensive pattern knowledge that includes a wide choice of patterns in various domains provides greater opportunities for practitioners to employ patterns to solve a wider range of problems, which would effectively increase pattern usage. Comprehensive pattern repositories, such as [Booch 2008], with relevant search and indexing facilities, would encourage more software engineers to employ patterns. As shown in this study, any increase in pattern usage is proportionally reflected in pattern usefulness, and conversely, pattern usefulness will encourage greater pattern usage. This relationship between the pattern usage and the usability and usefulness of patterns is important to leverage in enhancing software quality through greater use of patterns.
- **Provision of research on usefulness of patterns:** Evidence of the usefulness of patterns would also encourage practitioners to employ patterns because it has been indicated in the survey that as the rate of pattern usage increase, the pattern usefulness will also increase proportionally. It is therefore important that substantial research on the evaluation of utility and usefulness of patterns, such as this study, at both academia and industry be frequently conducted.

One of the issues with patterns, as indicated by the survey, is the minimal use of process-based patterns. The results show that pattern usage is overwhelmingly concentrated on design patterns and that process patterns usage is very low comparatively. The Figure 3-9 shows the proportion of companies using process patterns. Only six out of the 67 (8.9%) respondents used process patterns, and only one out of the 67 (1.5%) surveyed companies used patterns frequently. It therefore appears from the sample, that while many companies were satisfied that the employment of design patterns were useful in software development design and architecture, they were not convinced that the application of process patterns was beneficial to software development practice. This result prompted the main research topic of this study to investigate the utility of process patterns. One of the reasons for the low usage of process pattern in industry could be because formal development methodologies and processes are little understood and practiced in many immature software development companies. Some studies have shown that 35% of software development organisations have an ad hoc, individual-based, and informal development process in place [Yourdon 2008]. Evidence of positive effect of process patterns, as sought in this research, could encourage software organisations to use them in their development practice.

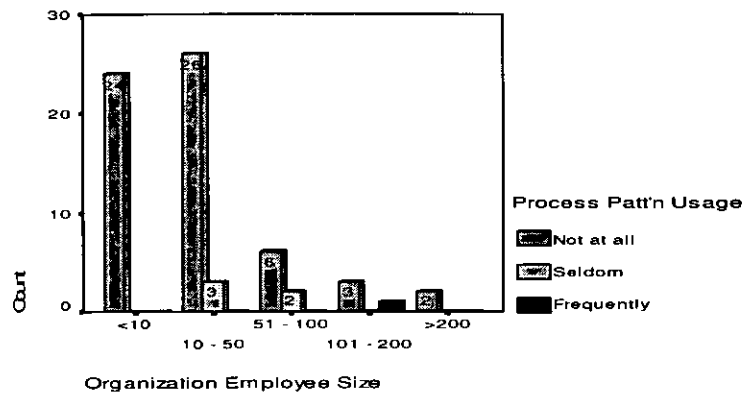


Figure 3-9 Process patterns usage

Companies that did not use patterns gave a number of reasons as to the rationale for not doing so. The results are depicted in Table 3-5. Eighty eight percent of the respondents gave 'lack of skilled staff', as a reason for not using patterns, which showed that such organisations felt that patterns required the expertise that they did not have within their development teams. Patterns are often hard for inexperienced practitioners to use properly and this appears to be one of the reasons hindering companies in using them. A substantial proportion of organisations (81%) felt that patterns are not required in their software development practice. These views were mostly expressed by small software development companies who were involved in developing small user-interface based web applications for which fewer patterns are available compared to larger and multi-tiered applications. Such companies therefore believe that patterns would not provide a significant advantage in their practice.

Reasons	Yes - Answers %	No - Answers %
Lack of Skilled Staff	88	12
Patterns outdate quickly	33	67
Patterns are not required	81	19
Patterns have side effects	29	71
Not Fully Reliable	22	78

Table 3-5 Reasons for not using patterns

The surveyed organisations were asked if they had plans to use patterns in the future. Ten out of the 27 (37%) that did not use patterns said that they had no plans, while 17 (63%) planned to use patterns in the next 12 months, as is illustrated in Figure 3-10. It shows that the majority of the companies that were not using patterns at the time, had decided that pattern usage would be beneficial to their practice, and were considering using them in the future.

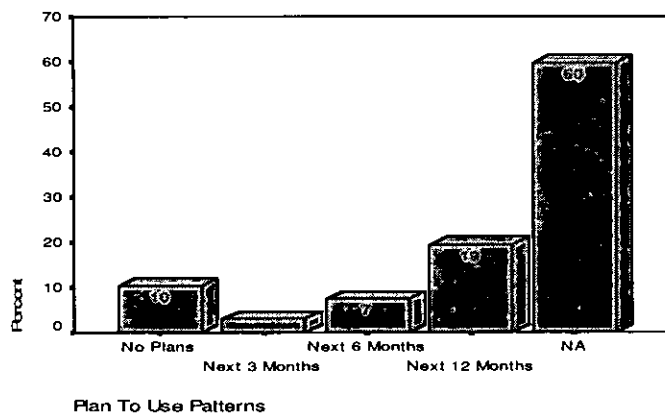


Figure 3-10 Companies planning to use patterns

### 3.3.5.2 Pattern Development

While the popularity and application of using patterns in software development practice is high and steadily growing [Buschmann 2007b], the survey showed that only a relatively small percentage of the software development organisations were engaged in developing patterns. As depicted in Figure 3-11 only four of the 67 respondents (6%) developed patterns. This indicates that while many companies utilise software patterns that have been published in the literature, very few are prepared to put in the effort to extract and write up patterns based on their own company-wide experience and practice and publish them. One reason for this is the lack of sufficient training in extracting and writing patterns [Manolescu et al. 2007]. There should therefore be a concerted effort by the pattern community to encourage, and provide support for software development organisations to engage seriously in producing and mining patterns. In particular, the pattern community should establish an authoritative pattern repository, with useful indexing facility, to which pattern users can both contribute and refer. The pattern community should further encourage software organisations to put in place training programmes on pattern development for their development teams. Training and practice, is immensely important in providing the necessary skills for development teams to be able to write quality patterns. The pattern community has introduced a shepherding process in which an experienced patterns write helps beginners to write patterns for publication. The shepherding mechanism should be encourage to be used and adopted in the software industry in increasing the number of engineers that have the skill to extract and write patterns. It should be borne in mind that badly written so-called patterns are worse than not producing patterns at all, as any solution that they provide could be misleading and therefore damaging to the applications that use them (see Section 2.3.2).

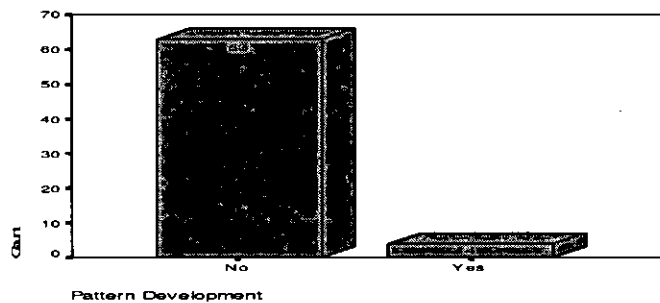


Figure 3-11 Companies developing patterns

### 3.3.5.3 Pattern's Effect on Software Quality

Participants' opinions were sought in the survey regarding the effect of patterns on a number of software quality attributes. The results are presented in Table 3-6.

	Use Patterns	Positive %	Negative %	Neutral %	Don't Know %
Testability	Yes	55.0	5.5	27.0	12.5
	No	3.7	0.0	22.0	74.0
	Total	34.4	3.3	25.0	37.4
Reusability	Yes	77.5	5.0	10.0	7.5
	No	7.4	0.0	3.7	88.9
	Total	49.2	2.9	7.5	40.3
Maintainability	Yes	57.5	7.5	20.0	15.0
	No	0.0	3.7	3.7	92.0
	Total	34.3	5.9	13.4	46.0
Portability	Yes	65.0	15.0	12.5	12.5
	No	0.0	3.7	3.7	92.0
	Total	38.8	10.4	9.0	44.5
Changeability	Yes	80.0	2.5	10.0	7.5
	No	7.4	3.7	3.7	85.0

	Total	50.7	3.0	7.5	38.7
Interoperability	Yes	82.5	2.5	12.5	2.5
	No	0.0	3.7	3.7	92.0
	Total	49.3	3.0	9.0	38.6
Efficiency	Yes	42.0	20.0	35.0	2.5
	No	3.7	0.0	7.4	88.9
	Total	26.6	11.9	23.9	37.3
Reliability	Yes	55.0	37.5	22.5	15.0
	No	0.0	0.0	7.4	92.0
	Total	32.8	22.4	16.4	46.0

Table 3-6 Participants' viewpoints on the effect of patterns on quality attributes

Reusability, changeability, and interoperability were the quality attributes that had the highest score of between 77 and 83%. The score shows that a substantial majority of the respondents that used patterns viewed these three attributes as the most influential benefits of software patterns. However, only 42% thought that the usage of patterns would improve efficiency, while 20% believed that in fact patterns had a negative effect on efficiency. The results on the reliability attribute were surprising. While 55% of the respondents that used patterns believed they had a positive effect on reliability, 37.5% thought that it had a negative effect, which is a surprisingly high proportion. This was rather unexpected because patterns, theoretically being proven solutions, should not be unreliable or have a negative effect on the reliability of the software that adopts them. The results indicate that while patterns may be reliable themselves, they do not necessarily enhance the reliability of the software that use them and could indeed in some cases decrease their reliability. It appears to suggest that patterns could influence aspects of a software development in a way that renders the resulting software less reliable. We recommend that further empirical/experimental investigations to be conducted to determine the effect of patterns on the reliability of the software that employ them. Such investigations could outline specific deficiencies and issues with patterns with regard to the reliability attribute and recommend strategies to resolve them.

The correlation between pattern usage and their effect on both reusability and maintainability were investigated which are presented in Figure 3-12, and Figure 3-13. There is a statistically significant and positive correlation between pattern usage and the reusability attribute with Corr. Coef.  $R=0.523$ , and Sig.  $p=0.001$ . There is also a statistically significant and positive correlation between pattern usage and the maintainability attribute with Corr. Coef.  $R=0.459$ , and Sig.  $p=0.007$ .

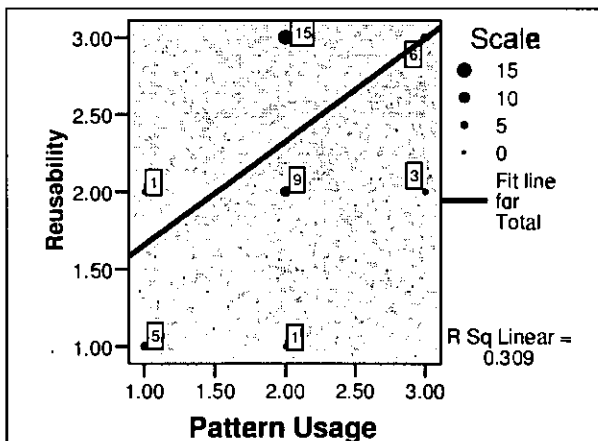


Figure 3-12 Correlation between reusability and pattern usage

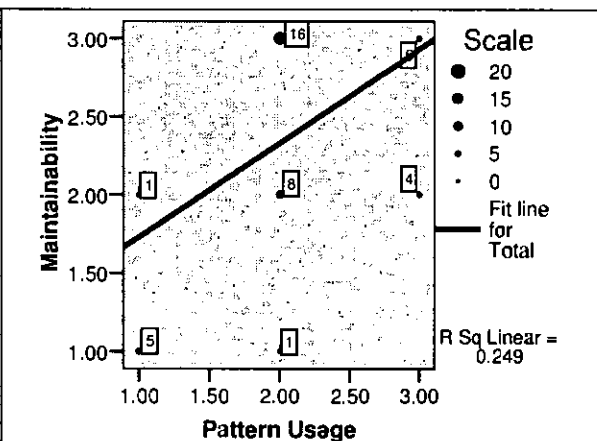


Figure 3-13 Correlation between maintainability and pattern usage

There was also found to be a statistically significant positive correlation between testability, reliability, and pattern usage as depicted in Figure 3-14 and Table 3-7. The results indicate that as pattern usage increases, testability and reliability of the application that implements them will also increase proportionally. This outlines a further benefit of pattern usage, which is to enhance both testability and reliability proportionally. The results

further show that there is a positive correlation between testability and reliability, indicating that any change in either attributes is proportionally reflected in the other.

		Pattern Usage	Reliability	Testability
Pattern Usage	Pearson Correlation	1	.465	.529
	Sig. (2-tailed)		.002	.000
	N	40	40	40
Reliability	Pearson Correlation	.465	1	.742
	Sig. (2-tailed)	.002		.000
	N	40	40	40
Testability	Pearson Correlation	.529	.742	1
	Sig. (2-tailed)	.000	.000	
	N	40	40	40

Table 3-7 Correlation analysis for testability, reliability, and pattern usage (significant at the 0.01 level)

**Correlation - Testability, Reliability, and Usage**

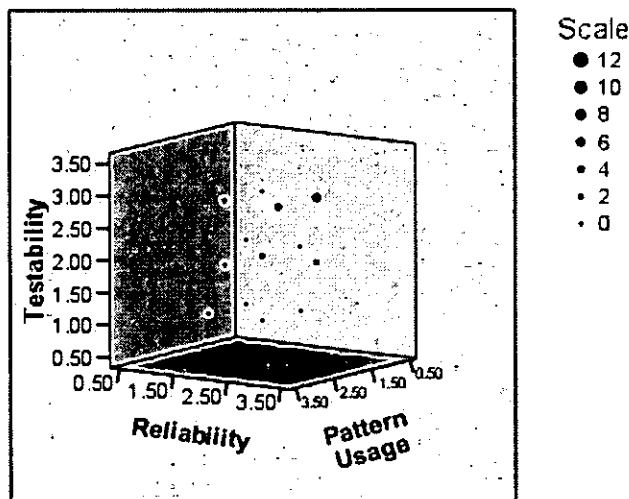


Figure 3-14 Correlation between pattern usage and testability-reliability quality attributes

The positive effect of pattern, on communication between team members, has been reported by some studies [Beck et al. 1996] [Hahsler 2005] [Unger and Tichy 2000]. The survey results, depicted in Table 3-8, corresponds to those findings in indicating that the majority of the surveyed participants believed that patterns improved communication between development team members.

Question (Pattern Users Only)	Yes %	No %	Doo't Know %
Do you believe patterns contribute towards better communication between software development team members	61	27	12

Table 3-8 Patterns' effect on communication

Further results are presented in the 'Appendix D. Results'.

---

### 3.4 Summary

This chapter discussed two preliminary surveys to evaluate the impact and usage of both architectural and software patterns. In the first survey, architects from 26 participating UK universities were asked for their viewpoints and teaching practices on patterns. The results of the survey showed that, 22 out of the 26 (84.6%) that responded did not teach patterns at any level. Out of the remaining four that taught patterns, two taught it at undergraduate levels, one at postgraduate and one at both undergraduate and postgraduate levels. There was found to be a positive correlation between pattern usage and architect's viewpoints. The survey also showed that the architects' viewpoints about patterns remained divided, but generally, support levels for patterns were shown to be low. While the majority of the surveyed architects did not favour patterns, some believed that it was an important concept that was timeless and was relevant now and in the future. Issues such as anti-creativity and unscientific aspects were amongst the main criticisms of the architectural patterns. The chapter discussed whether such issues also applied to software patterns.

In the second survey, a sample of software development companies were asked about their usage of patterns in their software development practices. Questions that were asked included whether they thought software patterns contributed towards software quality attributes such as, reliability, usability, and efficiency. The survey result indicated that 40 out of the 67 survey respondents (59.7%), used patterns in their software development practices. However, only four out of the 67 (i.e. 6%) companies that replied to the survey produced patterns. The results also indicated that process patterns were seldom employed. Only 6 out of 67 respondents said that they used process patterns (8.9%). There was also found to be a statistically significant and positive correlation between pattern usage and quality attributes such as reliability, testability, maintainability, and reliability.

The survey results indicated that, while design patterns were shown to be regularly used by software organisations, it appeared that process patterns were seldom used in the industry. The results prompted the main topic of this research to investigate the utility of process patterns through an experimental research method that involved software measurement. In the next chapter, therefore, software experimentation and measurement concepts will be discussed.

---

## Chapter 4 Software Experimentation and Measurement

---

### 4.1 Introduction

As briefly discussed in the introduction Chapter 1, software experimentation and software measurement are important topics and components of this research programme. Measurement is an essential element of the scientific process and includes such activities as measuring the variables to differentiate cases, measuring the changes in behaviour, and measuring the causes and effects. The key to the term software engineering is 'engineering' which intrinsically implies measurement and control. Grady [1992] stated, "Nothing should be accepted as software engineering unless it has been measured and proven". Demarco [1982] further noted, "You cannot control what you cannot measure" thus encapsulating the importance of measurement in software engineering. One area in which measurement is the essential component is in software experimentation. Software measurement and software experimentation are closely linked, as the experimentation process often involves measurement of some software attributes or entities. Experimental research methods can be employed in software engineering for many types of studies, one of which is to evaluate and validate new as well as established technologies and concepts (e.g. software patterns).

Based on the quantity and quality of software experimentations and measurements reported in the literature, neither software experimentation nor software measurement appear to have reached the maturity in software engineering that is enjoyed in other fields of science, such as physics, partly due to the relatively young age of just a few decades [Koziolek 2005] [Zelkowitz and Wallace 1998]. Our knowledge of software measurement is currently flawed to the extent that even international software measurement standards (i.e. ISO/IEC 15393), upon which practitioners often rely for support, have given misleading advice on software measurement [Kitchenham and Colin 2007]. As measurement is an essential component of experimentation, weaknesses in software measurement have a direct influence on the quality of experimentation. Such weaknesses have caused experimentation in software engineering to be a difficult and challenging undertaking and have therefore attracted fewer researchers resulting in lack of quality software experimentations [Tichy 1998].

Software measurement concepts and process is employed to measure and evaluate a number of attributes of software development projects through an experimental research method for assessing the utility and effect of process patterns. In this chapter, both software experimentation and software measurement in software engineering are reviewed including a discussion of software experimentation and measurement issues. The first section of this chapter refers to software measurement, where software measurement concept, software quality measurement, and software metrics are discussed. The second section discusses software experimental research, where there is a review of related works as well as a discussion of issues and difficulties in experimental research in software engineering.

### 4.2 Measurement Theory and Definition

Measurement theory is detailed and mathematically complex [Stevens 1946] [Torgerson 1958] [Campbell 1928] [Pfanzagl 1971]. Nonetheless, it is necessary that measurement in software engineering be based on sound theoretical and mathematical practice, to produce verifiable and valid results. Measurement theory deals with fundamental issues such as, the concept and meaning of measurement, the types of attributes that can and cannot be measured and their scales, the definition of measurement scales, meaningful measurement statements, the acceptable error margin, and whether what is measured is really the targeted attribute. Based on previous works on measurement theories, many have proposed measurement frameworks and principles for software measurement [Zuse 1998] [Fenton 1994] [Morasca and Briand 1997] [Fenton and Melton 1996]. It would appear however, that measurement theory constraints are too strict and have not therefore been used on building new measures but mostly used to analyse the properties of the existing ones [Morasca 2003]. Furthermore, many of the methods and theories proposed contain misrepresentation and flaws [Briand and Emam 1996] [Morasca et al. 1997b]. For example, some take issue with the notion that complexity metrics are additive, measurements fall into a number of distinct type scale levels, or that certain statistical techniques are not appropriate for some types of measures [Briand and Emam 1996]. Furthermore, many hard problems such as errors in modelling process and measurement process are not adequately addressed in software measurement theory [Shepperd and Ince 1993]. There is no established system of measurements in software engineering and therefore software engineers



often may need to consider techniques such as, rules of thumb, analogue conclusions and statements of trends, expertise, estimations, and predictions [Ebert and Dumke 2007]. Despite the outlined difficulties and flaws, the employment of the software measurement principles and guidelines is valuable in devising a validated measurement process.

Measurement is defined in a number of ways depending on where the emphasis and the focus of interest are placed. It is defined as “the process of empirical, objective, assignment of numbers to properties of objects or events of the real world in such a way as to describe them” [Finkelstein 1982]. It is further defined as “the process by which numbers or symbols are assigned to attributes of entities in the real world in such a way as to characterise them according to clearly defined rules” [Fenton and Pfleeger 1997]. In the latter definition, however, there are disagreements on the nature of the model/theory under which the rules are defined. While traditionalist argue that the ideal mode is the causal mode (i.e. change in the attribute causes a change in the value that will result from a measurement), our understanding of causal relationships for many variables are limited and so it would therefore, be impossible to discuss measurements of those variables in causal terms. It is for this reason the IEEE 1061 standard refers to correlation for validating a measure. Correlations however do not prove causal relationships and therefore this strategy is risk-prone [ibid]

Measurement is also defined as a mapping from the empirical world, to a more formal and mathematical world [Oman and Pfleeger 1997]. This mapping is depicted in Figure 4-1. The concept shared in all these definitions is that measurement is about the way numbers or symbols are assigned to entities to reflect a description or characterisation of an attribute. The values could represent a measure of the effectiveness of the development process or the quality of the products. Furthermore, such assignment of numbers is important in order to enable the differentiation and comparison of entities of interest. It should however be noted that assignment of symbols or entities such as vectors may prove problematic in some cases for the absence of ‘>’ or ‘<’ relations and may be only applicable to nominal scales. Based on the concepts and definitions outlined, for the purpose of this research the software measurement is defined as, the procedure of an empirical assignment of numbers, according to rules derived from a model or theory, to attributes of software engineering entities in order to describe them.

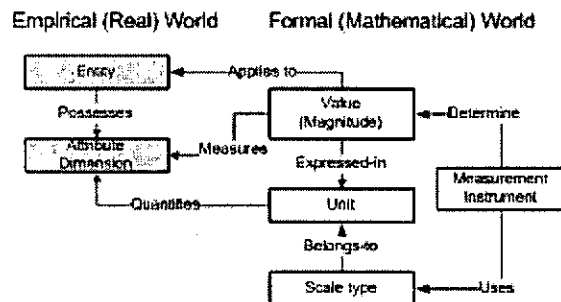


Figure 4-1 A model of measurement [Oman and Pfleeger 1997]

Although the terms measurements, measure, and metrics are often used interchangeably, they are in fact different in meaning. For the purpose of this research, the term measure is defined as ‘a number or symbol designating the value of a property of a software attribute’. A measure is the product of the measurement process. There is however some controversy about the usage of the term ‘metric’. The term was originally defined for the purpose of geometry mathematics (distance function [Hamming 1950]), and its usage in software measurement is problematic for being imprecise and perhaps misleading in some situations. While many authors (e.g. [Kitchenham and Mendes 2004] [Goodman 2004] [Grady 1994]) have used the term and continue to use it, some [Zuse 1998] [Whitmire 1997] have declined its usage. There are also efforts being made in homogenising the international standards to delete the word metric from the glossary of software measurement terms altogether. However the fact is that the term ‘metric’ is currently used in the software literature and will probably continue to be used until there is a universally agreed convention against its usage. Therefore, for the purpose of this research, metric is defined as “a quantitative measure of the degree to which a system, component, or process, possesses a given attribute.” based on the IEEE 610 definition. In practice a measure usually refers to lower level, more concrete measurement such as LOC (lines of code) and metric, to more abstract and higher level measurement usually derived from a measure, but there is typically overlap where either term can be used. An entity can be both a measure and a metric, depending on the context. For example, LOC as a number representing the number of lines of code is a *measure*; however, LOC as a way of measuring the number of lines

of code is a *metric*. It is clear that the use of both terms causes confusion and ambiguity and this is another reason for the argument to eradicate the term *metric* and the use the term *measure* in all cases.

Measurements are collected and analysed through a measurement process that includes the identification of the entity (e.g. a module) and its attributes of interest (e.g. size), followed by mapping the attributes to a mathematical representation (e.g. lines of code). Finally, the mathematical representation is interpreted in terms of their meaning in the empirical world (e.g. the size is too large, may need to be broken down). This is illustrated in Figure 4-2.

Having given a brief introduction to the measurement definition and theory, in the following section the purpose and benefits of software measurement are discussed.

### 4.3 Purpose and Benefits of Software Measurement

The role and importance of measurement in science cannot be overstated. Lord Kelvin [Thompson 1917] characterised this importance in stating that, “numerical accuracy is the soul of science”. While people find it necessary to understand many features of the empirical world (i.e. complexity of a software program), our brain is incapable of producing relevant empirical results from real world observations (empirical relational system) due to what Kriz [1988] calls intelligence barrier as depicted in Figure 4-2. It makes it therefore necessary to use such tools as numbers and symbols in mathematics and statistics (numerical relational system) to bypass the intelligence barrier, by properly translating empirical information to numerical objects and relations. The resulted numerical objects and relations can then be employed to improve the quality of software products.

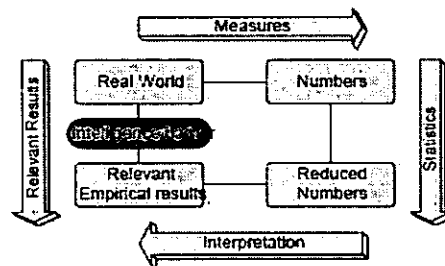


Figure 4-2 Measurement process and intelligence barrier [Kriz 1988]

Measurement is important for deriving the basis for estimation, quality control and prediction as well as to provide help with many activities such as tracking project progress, determining relative complexity, analysing defects, and experimentally validating best practices [Grady 1994]. Measures (e.g. effort, cost, duration, faults, failures, and changes) are valuable for understanding and improving the software development processes as they define targets to aim for in developing high quality software [Fenton and Pfleeger 1991]. In an empirical study, Hall et al. [2001] found that both managers and developers viewed many aspects of measurement beneficial to the software projects. In particular, they viewed the tracking of progress, improvement of planning and estimation, and identification of specific problems to be the major benefit of measurement.

A key benefit of measurement, directly related to the topic of this research (i.e. process patterns), is in process improvement. Development processes are improved through continuous quality assessment [Sommerville 2007]. It is through attempts in improving software development processes that process patterns are often formed and established. When a process activity is matured through measurement and quality assessment and repeatedly produces workable and proven solution in different applicable circumstances, it becomes a ‘process pattern’ in practice. It then needs to be written up in accordance with the structural and contextual requirements of pattern and be offered for publication as a process pattern. This is illustrated in Figure 4-3.

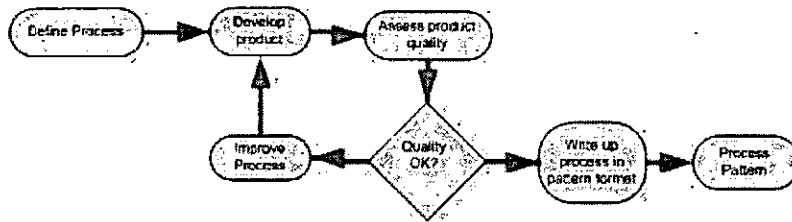


Figure 4-3 Process pattern development through process improvement

Software measurement provides the means of deriving a numeric value for an attribute of a software product or process that facilitate objective comparisons between techniques and processes. This is the context in which measurement is employed in this research. The measurement process is used to compare the quality of a number of software product and process attributes in development processes that use 'process patterns' and those that do not. The methods and context in which the devised measurement process is used in the experimental research method are discussed in detail in Chapter 5, and Chapter 6.

Measurement scales provide the principles and the yardstick on which measurements can be based. There are many types of measurement scales, which will be discussed in the following section.

#### 4.4 Measurement Scales

Understanding the nature and scales of collected data is important for operations such as statistical tests, aggregations, and correlations of the variables concerned. It is therefore important to determine the measurement representation most suitable for the attribute to be measured. Stevens [1946] proposed the following four levels of measurement which is widely adopted and used in software measurement: 1) nominal, 2) ordinal, 3) interval, and 4) ratio. In addition to Steven's four classes, a further scale called 'absolute scale' was also proposed. These scales are listed in Table 4-I.

The *nominal scale* is the simplest scale and it only places the entities in different classifications. The classes are identified by unique symbols, or numbers, and cannot be interpreted as anything other than identifiers. The only comparisons that can be made between variable values are equality and inequality. There are no 'less than' or 'greater than' relations among the classifying names, nor operations such as addition or subtraction. The *ordinal scale* is used if the task is to order members of a group according to the extent to which they possess the chosen attribute. Comparisons of greater and less can be made, in addition to equality and inequality. However, operations such as conventional addition and subtraction are still meaningless. The *interval scale* allows the magnitude of the attribute to be expressed numerically, as a distance from some chosen point of reference. In this scale, the differences between arbitrary pairs of measurements can be meaningfully compared and operations such as addition and subtraction are therefore meaningful. The zero point on this scale is arbitrary and negative values can be used. The *ratio scale* expresses the magnitude of the measure as a multiple of a chosen unit of measurement. It preserves the ordering and the size of the intervals and the ratios between entities, and therefore operations such as multiplication and division are meaningful. The *absolute scale* is used for counting and only uses rational numbers. In contrast to the other four measurement scales, the absolute scale is not transferable. That is, the scale is unique and cannot be rescaled. For example, while in non-absolute scales results of a classification expressed in a system of pictorial symbols can be mapped into a system of colours, the alphabet, or the set of natural and rational numbers, such transformation is not possible in the case of the absolute scale.

Scale Type	Admissible Scaling Transformations	Defining Relations	Application Examples
Nominal	$M'=f(M)$	Equivalence	Name of programming languages
Ordinal	$M'=f(M)$ ; If $M(A1) \geq M(A2)$ Then $M'(A1) \geq M'(A2)$	Equivalence Greater than smaller than	A ranking of failures (severity)
Interval	$M'=aM+b, a>0$	Equivalence, Greater than/smaller than, Relative scale values	Beginning date, End date of activities (as measures of time)
Ratio	$M'=aM, a>0$	Equivalence, Greater than/smaller than, Relative scale values, Ratio between value	LOC (as a measure for program size)
Absolute	$M'=M$	Equivalence, Greater than smaller than, Relative scale values, Ratio between scale values, Absolute scale values	the number of occurrences of something

Table 4-1 Measurement scale types

The categorisation of data into the scales described above suggests restrictions in the type of statistical analysis that can be applied to each scale classification (e.g. parametric-test can only be applied to the interval and ratio scales). For example, it is not meaningful to establish a statistical mean over an ordinal-type measurement, since that assumes a constant interval between all the points of the scale. However, while the distinction between categorical and continuous data is important, the rigid application of Steven's four measurement scales is not necessary [Dewberry 2004]. Often this classification is too restrictive to apply to real world data and often lead to degrading data by rank ordering and unnecessarily using nonparametric methods [Velleman and Wilkinson 1993] [Briand and Emam1996]. Furthermore, strict application of this taxonomy would substantially hinder the progress of empirical research in software engineering. The experiment data in this research project were of type interval or ratio (these are also referred to as continuous data) and parametric tests were used for their statistical analysis as both these measurement scales are suitable for parametric tests even by strict adherence to Steven's [1946] principles.

Knowing and understanding techniques for measuring software attributes is important in using an appropriate technique for measuring a particular attribute. In the following section, these techniques are presented and discussed.

## 4.5 Measurement Techniques

While some software attributes can be measured directly, many can only be measured indirectly. In order to measure such attributes, indirect measurement techniques are employed. In this research, both types of measurements were used. These are discussed in the following section.

### 4.5.1 Direct and Indirect Measurement

*Direct measurement* refers to measurement of an attribute when no other attribute has a direct or indirect influence. A direct measure is defined as "a measure that does not depend upon other attributes." [IEEE STD 1061]. Direct measurements are used to measure internal attributes. Examples of direct measures used in this study are, no. of lines of code (LOC), test duration (time in hours), and defects discovered in testing.

*Indirect (or Derived) measurements* are used when an attribute can only be measured in relation to other attributes. They are used to measure external attributes and may be necessary where temporal considerations prevent direct measurement. Indirect (or derived) measures often demonstrate the interactions and relationships between direct measures and are often a factor or function of a number of direct measures. Examples of indirect measurements are, defect density, productivity, and test effectiveness. Table 4-2 below depicts a number of examples of indirect measurements and the way they can be measured using direct measures. The Table 4-3 depicts internal and external attributes for software development products, which can be measured by direct or indirect measurement techniques.

Indirect (Derived) Measures	Evaluation using direct measures
Programmer Productivity	(LOC produced) / (effort). [This is a widely used, but controversial method of calculating productivity partly due to difficulties in defining and measuring Lines of Code consistently]. Function points are also used instead of LOC.
Defect Density	(No. of defects) / (size)
Defect Detection Efficiency	(No. of defects detected) / (Total No. of defects)
Requirements Stability	(No. of initial requirements) / (total No. of Requirements)
Test Coverage	(No. of items covered) / (total No. of items)
System Spoilage	(effort spent fixing faults) / (Total project efforts)

Table 4-2 Examples of indirect measures

Products	Internal Attribute	External Attribute
Specification	Size, re-use, modularity, redundancy, syntactic correctness.	Comprehensibility, maintainability
Design	Size, re-use, modularity, coupling, cohesiveness, functionality.	Complexity, maintainability
Code	Size, re-use, modularity, coupling, functionality, algorithm complexity, structure	Reliability, usability, maintainability
Test data	Size, coverage, level	Comprehensibility

Table 4-3 Examples of Internal and external attributes for products

An important aspect of indirect measure is that they should not exhibit unexpected discontinuities. For example, in the definition of measurement M1, defined as  $M1 = x/(y-1)$ , the measurement M1 is undefined and invalid if y were to be 'one'. This measure therefore would be valid for conditions under which y would never have the value 'one'.

Software metrics are the essential component of a measurement process. They represent a method or formula in measuring a software attribute. There are different types of software metrics, which will be discussed in the following section.

## 4.6 Software Metrics

Software metrics deal with the measurement of the software product and the process through which it is developed. They are numerical measures of a product or process that is part of a software project. There are however, difficulties in formalising standardised metrics for many software attributes that are generally and entirely accepted by the software community as a whole. For example, although in the past three decades, there have been many attempts to develop a single metric to provide a comprehensive measure of software complexity, no one measure has been developed around which a consensus has been achieved [Pressman 2005]. The problem is that there are many different views of what constitutes software complexity and what attributes of a system lead to it. There are therefore no standardised and universally agreed and applicable software metrics on complexity and other attributes [Sommerville 2007].

Many characteristics and qualities are suggested by software practitioners and authors for a metric, some of which are difficult to achieve in practice. For example Mills [1998] states that good metrics should be simple and precisely definable, objective as far as possible, valid, and robust. Ince et al. [1993] argue that for a metric to be truly useful it should be measurable (i.e. be based on facts), independent (i.e. changes in its value does not effect quality of software), accountable (i.e. contain detailed on how and when the metric was measured) and precise (i.e. has known level of tolerance). Furthermore, Basili et al. [1996] also recommend that for maximum utility in analytic studies and statistical analyses metrics should have data values that belong to appropriate measurement scales (see Section 4.4 Measurement Scales). While such desired attributes of a metric is something to aim for in any measurement process, it is a challenging endeavour, which often proves hard to achieve in practice. Measurement challenges are further discussed in the Section 4.10.

### 4.6.1 Process and Product Metrics

Product and process metrics are the two main types of software metrics both of which were used in this study. While the product metrics measure the attributes of the software products, process metrics measure the attributes of the process employed to obtain the results. Process metrics are therefore used to measure attributes of a

software development process. They measure process attributes such as, 'Number of defects introduced per developer hour' or 'Number of changes to requirements'. The most popular and referenced process metric types include management support metrics, productivity metrics, efficiency metrics, process quality metrics, actual vs. planned metrics, and traceability metrics [Grady 1992] [Humphrey 1989]. Product metrics are employed to measure attributes of the software itself such as size or complexity. Products include any artefact or document, such as prototypes, test harnesses, specification documents, that is produced during the life of software. Some examples of more commonly used product metrics are: Lines of Code, Function Points, and Cyclomatic Complexity.

Software metrics can be categorised into four main groups in accordance with the four major activities of a development life cycle [Pressman and Ince 2000]. These are as follows:

- Metrics for the analysis model
- Metrics for design model
- Metrics for source code and implementation
- Metrics for testing

This classification of metrics reflects the metrics strategy designed for this study. The metrics employed in this study through the measurement process fall into these categories. The measurement process designed for this study is discussed in detail Chapter 6.

#### 4.6.2 Composite/Hybrid Metrics

Composite/hybrid metrics are created by aggregating several resource (or other composite) metrics according to a specific algorithm, such as averaging one or more metrics over a specific amount of time or by breaking them down according to specific criteria [Keller and Ludwig 2002].

The main advantage of using composite/hybrid metrics is that they enable measurement to be more generalised and to represent a multitude of factors that affect the quality criteria. Using a number of metrics in aggregation, where each individual metric probes a different aspect of an attribute, may give a more valid and precise measure of the overall quality of the attribute. This is to offset some of the inadequacy of single metrics to satisfy many measurement objectives. Single metrics are not sensitive to problems of quality factor trade-offs [Shepperd 1996] and a single metric is seldom adequate to encapsulate properties of interest [Basili and Rombach 1988]. Furthermore, single metrics in isolation are too simplistic to provide adequate explanation for software engineering phenomena [Shepperd and Ince 1993]. In order to overcome such weaknesses, some authors have proposed Composite/Hybrid metrics by combining the best aspects of existing metrics. Harrison and Magel [1981] have shown that neither Halstead's [1977] nor McCabe's [1976] complexity metric is sufficient individually, and a combination of the two would produce a better metric. A varied combination of metrics in hybrid format has also been used by Munson and Khoshgoftaar [1990, 1992] in the form of relative complexity metrics. Kitchenham and Mendes [2004] also used composite metrics to propose a method of productivity measurement. Composite measures composed of multiple measures can further help reduce measurement errors (i.e. random errors, method variance), since single measures are contaminated by irrelevant aspects (e.g. extraneous variables) of methods used [Campbell and Fiske 1959].

There are however two issues with composite metrics that need careful attention and consideration:

1. Aggregation (e.g. averaging) of a number of primitive metrics could make the resulting composite metric less sensitive [Melton 1990]. Therefore, metrics have to be aggregated with careful consideration not to adversely affect the sensibility of the resulting composite metrics. However, a little loss of sensitivity may be acceptable if the resulting composite metric provides other benefits such as simplicity and generality.
2. Conflict of scales and dimensional inconsistency in compositing the metrics could invalidate the resulting metric. For example, one has to be careful that metrics of different units are not inappropriately aggregated.

Provided that those two points are taken into consideration, the inclusion of composite/hybrid metrics could be beneficial and valuable in many measurement programmes. The use of composite metrics in this research was considered. However, a decision was made not to use them since that would unnecessarily complicate the measurement process and would run the risk of producing less sensitive results.

An important element of a measurement process is the validation of the metrics used in the measurement process. Having discussed the metrics in the previous section, in the following section the validation of metrics is discussed.

## 4.7 Measurement Validation

A fundamental concept of measurement is that the measurement of an entity must not presume the measurement of related entities other than the one being measured. A full understanding of what is being measured is important. In particular, attention should be paid to the nature and scales of data collected, to ensure that any aggregation and correlations of the data and variables are valid and meaningful.

Validation of a software measure is the process of ensuring that the measure is a proper numerical characterisation of the claimed attribute [Baker et al. 1990]. Often, however, it is assumed mistakenly that a software measure is only valid if it can be shown to be an accurate predictor of some software attribute of general interest like cost or reliability. This in fact is only true for the validation of a prediction system, which is defined as “the usual empirical process, of establishing the accuracy of the prediction system in a given environment, by empirical means” (i.e., by comparing model performance with known data points in a given environment) [ibid]. There is, therefore, a difference between validation of a measure and validation of a prediction system. Crucially a measure is not always part of a prediction system (e.g. program size used to predict project effort).

A key attribute of a good measurement process is the quality of its validation considerations. Fenton and Melon [1996] suggest two most important questions to ask in validating a measure, 1) how much do we know about the attribute to be measured, and 2) how do we know that we are measuring the attribute we want to measure. These two questions refer to what is called, “construct validity” and are the basis for major criticisms of software metrics [Nance and Arthur 2002]. There is a negative correlation between a measure and its underlying attribute as there will be more distortion when a measure is less tightly linked to its underlying attribute [Ebert and Dumke 2007]. In order for a measure to be validated, the following four validity checks are necessary [Kitchenham and Pfleeger 1995]:

- **Attribute validity:** Interested attribute (both directly and indirectly measurable) is actually exhibited by the entity to be measured.
- **Unit validity:** Employed measurement unit is an appropriate means of measuring the attribute.
- **Instrument validity:** Any model underlying a measuring instrument is valid and the measuring instrument is properly calibrated.
- **Protocol validity:** An acceptable measurement protocol is adopted.

Generally, there are a number of questions that need to be addressed in order to validate a metric. These include [Kaner and Pond 2004]:

- 1) What is the purpose of this measure?
- 2) What is the scope of this measure?
- 3) What attribute are we trying to measure?
- 4) What is the natural scale of the attribute we are trying to measure?
- 5) What is the natural scale for this metric?

Although validity consideration is an important component of a high quality measurement process, it is often missing in published measurement-related research [Koziolek 2005]. While a complete validation process both theoretical and empirical is often difficult to implement in practice, essential validation checks such as construct validity should be part of any measurement process. Continuous and systematic inclusions of such validity routines in software measurement process not only enhances the probability of achieving validated measures, but also generally helps in moving software engineering forward towards a more quantifiable and mature discipline. In defining the measurement process for this study the validity issues discussed in this section were considered and implemented wherever applicable.

In the above section, the measurement principles and components such as metrics with regards to this research were discussed. One important utility of metrics is their application in quality measurement and evaluation. This is the context in which metrics were employed in this research. In the following section, therefore, quality measurement techniques and issues are discussed.

## 4.8 Software Quality Measurement

The main goal of software measurement is to improve software quality. As software is becoming increasingly complex and critical due to increasing business demands for more sophisticated software, the quality of software products is a major concern for both software producers and users [Fuggetta 1998]. Software quality is probably the most desired and sought after goal in software engineering which has so far been unattained [Blaine and Cleland-Huang 2008]. While there have been improvements in the quality of software over the last couple of decades, partly due to the advent of object-oriented development and the associated CASE support, software quality measurement continues to be a challenging endeavour. Even after using the software for a long period, it is difficult to measure software quality attributes such as maintainability [Sommerville 2007]. While the term, 'quality' in software engineering might seem self-explanatory, there are many different views of what is meant by quality and how it should be measured or assessed. The term, 'software quality' denotes an elusive and multidimensional concept [Gillies 1997], and software quality attributes are often in a conflicting relationship to one another. For example, it may be that the application of a design pattern results in code that is, more flexible, but more complex as well. Furthermore, although an increasing number of software quality standards emphasise the need for measurement (ISO, SEI, and IEEE), most provide little detail as to what exactly should be measured and how the results should be used in the assessment of software quality. The fundamental issue is that our understanding of software quality and its measurement is not substantial [Oman and Pfleeger 1997]. While characteristics such as, fit for purpose, conformance to specification, degree of excellence, and timeliness are often proposed for software quality, the problem however is that such characteristics and definitions are not much use in offering the ability to quantify and measure quality. There is also an argument as to whether the quality of the development process affects the quality of the delivered product directly. While the relationship between process quality and product quality in software engineering is complex and consequently (i.e. one cannot predict how process change will influence the product), experience has shown that process quality has a significant effect on the quality of the software [Sommerville 2007].

Software constitutes a component of a larger system in which other components and factors such as humans, other products, and hardware are involved. Therefore, the whole-system characteristics influence the criteria for software quality. Software quality is therefore difficult to define universally and unambiguously, resulting in the quality measurement to be often a subjective task. There have been a number of proposed ways of measuring software quality [Basili 1995] [Kitchenham and Pfleeger 1996] [Kitchenham and Mendes 2004]. One suggested method is to investigate the 'ilities' (non-functional requirements) of the system in attributes such as stability, maintainability, reliability, verifiability, portability or extendibility. However, measurement of such quality attributes and their exact operational definition is the subject of much argument and disagreement. Furthermore, there is no way of directly measuring such quality indicators. Another method is to investigate program correctness through looking at the defect rate or defect density (i.e. defects / LOC). It is also possible to estimate the number of defects that remain in a piece of code, when it is completed through a method called the 'latent defect rate'. However, the problem with this method of measuring quality is that, while it accounts for defects, it does not cover other quality attributes essential to a quality software product. Furthermore, as the number of defects detected is dependent upon the quality of inspection process [Schach 2005], the number of defects recorded may not represent the actual number of defects in the software. The overall software quality can also be measured as, weighted linear combination of a number of quality attributes such as reliability, performance, security, fault tolerance, testability, and maintainability (i.e.  $Q = w_1R + w_2P + \dots$ ) as proposed by Voas and Agresti [2004]. The problem with this proposal is however, the difficulty in accurately measuring the constituent software attributes. In this project, attributes such as defect density are used to gauge the quality of the software projects that use process patterns in comparison to those that do not. Software quality measurement is currently imperfect, and metrics can only provide indications of quality rather than answers with absolute certainty. The more important aspect of metrics is their benefit in developing a measurement culture within the software development organisations through their continuous application in software development projects.

Basili [2005] identifies several quality measurement methodologies proposed in the literature. They can be broadly categorised into the following two types: (a) Factor/Criteria/Metric model, and (b) goal-oriented models (e.g. Goal/Questions/Metric model - GQM). These are discussed in the following section.

### 4.8.1 Factor Criteria Metric Models (FCM)

The FMC models are the oldest (three decades) and the most well known quality measurement models and have been used in many commercial applications worldwide. These models are tree-like, where higher branches hold high-level quality factors such as reliability and maintainability. The quality factors themselves are composed of lower level criteria, such as the structured and conciseness, which are easier to understand than the factors. For



these criteria then the actual metrics are proposed. FCM models are depicted in Figure 4-4, Figure 4-5, and Figure 4-6.

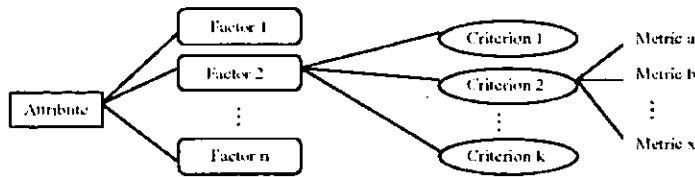


Figure 4-4 Factor-Criteria-Metrics general model

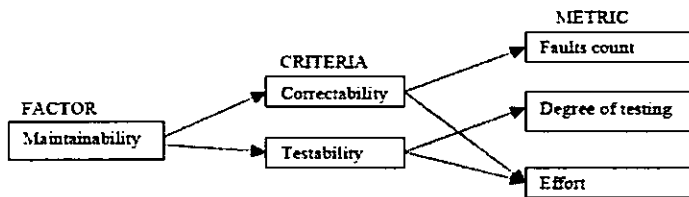


Figure 4-5 An example of FCM model for maintainability

An implementation of the FCM quality model is proposed by McCall [1977]. It incorporates a number of criteria in three major categories: product operation, product revision, and product transition (Figure 4-6). The model proposes a set of factors that affect software quality which are known as the McCall factors. These factors are based on three important aspects of software products, namely, operational characteristics, changeability, and adaptability.

Although widely used, the FCM model has some drawbacks and limitations. These include, 1) mapping the criteria onto the metric is obscure, and 2) there is poor capacity for mapping quality problems to causes [Marinescu 2004]. The criteria/metric mapping is “hidden” behind the arrows that link the quality criteria to the metrics, making it impossible in most cases to trace back and determine the rules and principles that dictate the mapping. Furthermore, the FCM does not help in finding the real causes of the detected quality flaws, because abnormal metric values indicate the symptoms of a design or implementation problem and not the problem itself. A treatment can only be advised or applied when the problem, not only a set of symptoms, is known.

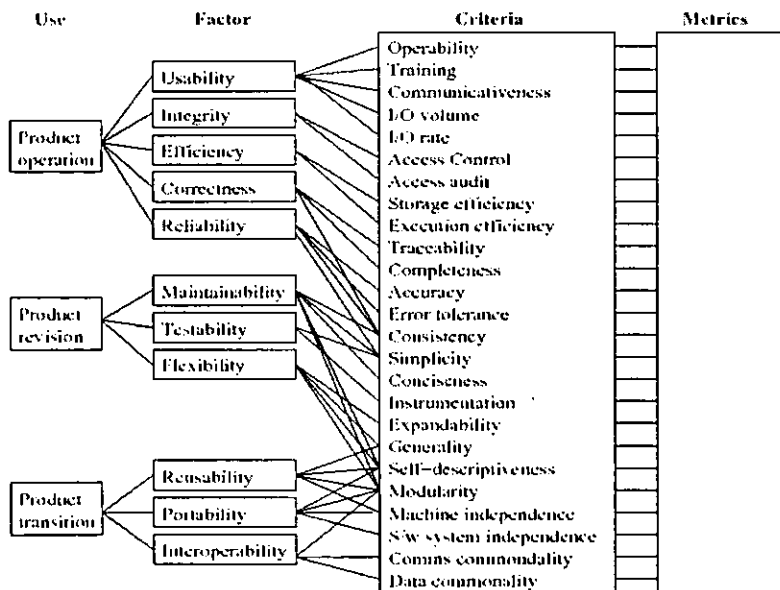


Figure 4-6 Factor/Criteria/Metrics model (McCall/Boehm model)

## 4.8.2 Goal Question Metric Model

The Goal/Question/Metric (GQM) model is a mechanism that provides a framework for developing a metrics programme. The approach was originally defined for evaluating defects for a set of projects for NASA. The application initially involved a set of case study experiments [Basili and Weiss 1984], but was later expanded to include various types of experimental approaches [Basili and Rombach 1988]. Goal-oriented measurement provides a strategy for deriving measures from measurement goals, to ensure the consistency and completeness of a measurement plan. The paradigm does not provide specific goals, but rather a framework for stating goals and refining them into questions to provide a specification for the data needed to help achieve the goals. The GQM as a goal-oriented measurement paradigm helps with the following tasks [Basili et al. 1994] [Basili 2005]:

- Ensure adequacy, consistency, and completeness of the measurement plan and therefore of data collection.
- Manage the complexity of the measurement programme
- Stimulate a structured discussion and promote consensus about measurement and improvement goals

One of the important aspects of GQM is that it forces problem definition and defines the metrics required to address them. Furthermore, as well as being flexible and applicable to almost any software measurement environment, GQM provides a context to understand metrics in addition to evaluating them by posing the question to which a metric is aimed to provide an answer [Shepperd and Ince 1993]. GQM is however software project centric and some have criticised it [Roche 1994] for being deficient in properly addressing the alignment between the technical and business objectives. The GQM paradigm consists of three steps: 1) Generate goals, 2) Derive related questions, and 3) Develop appropriate metrics. The structure is hierarchical as depicted in Figure 4-7.

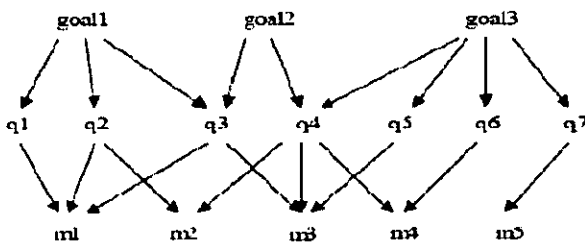


Figure 4-7 The Goal Question Metric Model

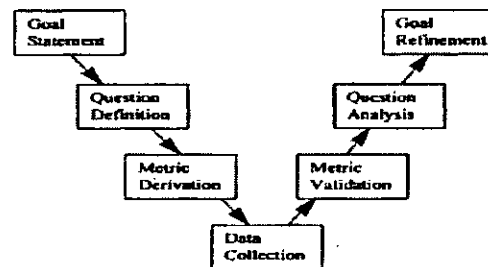


Figure 4-8 V-GQM Model

A weakness of the GQM model is that it is a standalone implementation and does not take into consideration previous implementations for validation and other purposes. In addressing this weakness, some have proposed that in addition to the top down approach, there should also be a bottom up procedure enhancing the usability and scope of the GQM paradigm [Hefner 1995]. Olsson [2001] also proposed a useful extension to GQM in which previous GQM implementations are studied, and lessons learned are used as feedback to the current or future GQM projects. This model is referred to as V-GQM and is depicted in Figure 4-8. Another weakness is the difficulty for GQM users to link measurement goals to higher-level organisational goals. This is important in providing the justification for an introduction of a measurement process. However, some have also been proposed extensions to GQM to address this issue [Basili et al. 2007].

Based on GQM, the Software Assurance Technology Centre (SATC) at NASA [NASA SATC][Wilson 1997] [Rosenberg 1996] developed a software quality metrics programme that covers risk management and quality assessment of the process and products of software development projects. The SATC model for metrics programme is depicted in Figure 4-9.

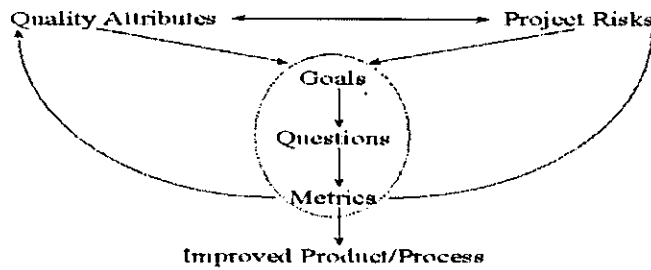


Figure 4-9 SATC Model for Software Metrics Programme

The SATC model defines a set of goals covering a complete development lifecycle. The defined goals are then associated to the software product and process attributes, for which a set of metrics is developed for their measurement. There are four goals defined for this, which are requirements quality, product quality, testing quality, and Implementation quality. In the implemented experimentation and measurement model for this research the GQM paradigm is used to develop a number of metrics to be used in the experimentation of this research. This is discussed in Chapter 6.

While the GQM paradigm offers a measurement model that is an improvement on the FCM model, the process is not repeatable (i.e. people may refine goals differently and therefore reach different questions and metrics each time). This is not, however, a significant weakness of the GQM paradigm since teams generate goals and the related questions and metrics according to their understanding of the circumstances. This inevitably means that different teams would generate different questions and metrics, which would not necessarily be a disadvantage. A further characteristic of GQM model is that it is not always clear when to stop generating questions and begin defining metrics. In other words, the granularity levels of the questions are left at the discretion of the teams implementing GQM. This is, in a way advantageous, since defining granularity levels for the GQM would make the paradigm too rigid, specific, and unsuitable to be applicable to all situations and circumstances for which it is intended.

The quality evaluation for this research is done through a measurement process, in which a number of metrics were selected to evaluate some quality attributes of the investigated software projects. Due to the benefits outlined above and the unique nature of the measurement programme, the GQM model was adopted as the measurement process strategy. A measurement process was devised and conducted based on GQM model. The detail of the devised GQM programme is discussed in detail in Chapter 6.

Many aspects of object-oriented (O-O) software are different to the classical software. As such, in many cases, there are different ways of measuring object-oriented software attributes. In the following section, the measurement techniques of object-oriented software are discussed.

## 4.9 Measurement of Object-Oriented Software

With the establishment of a popular programming paradigm called object-oriented programming, many researchers worked on providing metrics appropriate for the measurement of object-oriented applications and projects. The Object-Oriented approach uses concepts such as localisation, encapsulation, information hiding, inheritance, object abstraction, and polymorphism, making the software design and structure different to procedural programming. While many have proposed useful (O-O) metrics (e.g. [Lorenz and Kidd 1994], [Abreu 1995]), the most influential and important work in the field of object-oriented measurement is produced by Chidamber and Kemerer [1994], referred to as the CK metrics. The paper proposed six class-based metrics (suite of metrics) to measure software design attributes such as complexity and efficiency indirectly (Table 4-4).

Metric	Definition/Description
Weighted methods per class (WMC)	Sum of weighted methods per class
Number of children (NOC)	Number of immediate subclasses
Depth of inheritance Tree (DIT)	Maximum length from the node to the root of the tree
Coupling between object classes (CBO)	Count of classes to which this class is coupled
Response for a class (RFC)	Number of methods in the set of all methods that can be invoked in response to a message sent to an object of a class
Lack of cohesion in methods (LCOM)	The number of different methods within a class that reference a given instance variable

Table 4-4 CK metrics

The WMC is used in this study as a measure of complexity. It is measured as:

$$WMC = \sum_{i=1}^n C_i$$

Equation 4-1 Weighted Method per Class

Where  $C_i$  represent the complexity of method ( $i$ ). The complexity of each method is often measured by the cyclomatic complexity method [McCabe 1976], which is a count of the number of linearly independent paths through the source code. A further method of assigning a weighted complexity measure of 'one' to each method is also widely used. Chidamber and Kemerer [1994] who proposed the metric did not specify which complexity method should be used other than saying that it should have the properties of the interval scale. Churcher and Sheppard [1995] have found that assigning a weighted complexity measure of '1' to each method is as good an approach as using the cyclomatic complexity. This is the technique used to calculate WMC in the study.

The CK metrics, primarily applied to the concepts of classes, coupling and inheritance, were based on theoretical foundations and do not suffer as much from the criticisms made of previously published OO metrics. Many studies have reported positive results on the usefulness of CK metrics. It has been found that there is positive correlation between the 'depth of inheritance tree' (DIT) metric and the number of user-reported problems [Pant 1996]. Furthermore, the CK metrics were shown to be more effective predictor of fault proneness than extant code metrics [Basili et al. 1996]. In a comparison of three metric suites, Olague et al. [2007] showed CK metrics were better and more reliable predictors of fault-proneness than the MOOD [Abreu 1995] or QMOOD [Bansiya and Davis 2002] metrics.

However, a weakness of the CK metrics is that they produce rather poor size and effort estimations and they mostly concentrate on the application design. Some of the CK metrics are also criticised for failing to be based on the empirical relation systems, and adhere to representational conditions [Hitz and Montazeri 1996]. Nonetheless, it is widely accepted that CK metrics suite provide the foundation for OO measurement. Over the years, much research has been carried out on the validation and extension of CK metrics [Subramanyam 2003] [Basili et al. 1996] [Zhou 2006].

There are currently many difficulties and issues associated with software measurement. Having discussed important aspects of software measurement in the above sections, in the following section the difficulties, challenges, and issues with software measurement are discussed.

## 4.10 Software Measurement Issues and Challenges

Software measurement is a challenging but important component of a highly capable software engineering culture [Wieggers 1999]. Although measurement plays a central role in mainstream engineering disciplines, its role in software engineering is currently far less prominent. Only one third of all software engineering companies systematically employ techniques to measure their products and development projects [Meta 2002] [CIO 2003] [IQPC 2003]. Both practitioners and researchers are instructed to use measurement in software development and experimentation. There is however little concrete guidance about exactly how to start, and what has proven most effective in actual use [Oman and Pfleeger 1997]. Furthermore, measurements are done infrequently, inconsistently and incompletely and it is often unclear how the results were obtained, how experiments were designed and executed, and which entities were measured and how [Fenton and Pfleeger 1991].

One difficulty with software measurements is that they are often subjective rather than objective, which means that they are dependent on the environment in which they are made (e.g. the person(s) doing the measurement, location, and circumstances). In addition, while there are many ways that software attributes can be measured (e.g., size can be measured in lines of code, function points, tokens etc.), there are no industry wide standards governing which metric to use [Pressman 2005].

There are currently a small proportion of software organisations, which have an established and successful software measurement programme [Kaner and Bond 2004]. Only 20% of the organisations that implemented measurement programmes stated that it led to advancements and increased the bottom line [Dekkers 1999]. The intensive use of a single measure and, the use of too many measures are two of the top ten problems leading to failure in the implementation of software measurement programs [Rubin 1996]. Many of those that do have a measurement programme in place, have done so only to conform to criteria established in the standards such as the Capability Maturity Model [Fenton 1999]. There may be many reasons for such resistance to measurement programmes, one of which is the high costs involved in putting in place a comprehensive measurement programme. Some studies have estimated this to be between 3 to 6% of the overall cost of development of a software development project [Jones 1996] [Fenton 1999]. However, it is interesting that while the costs of introducing a measurement programme can be around 1% of R&D [Ebert et al. 2005], studies have shown that savings of as much as 10 – 20% on R&D can be made as a result [Kutz 2003]. One other reason for the lack of interest in implementing a measurement programme is the possible disadvantages and damaging side effects of such programmes. Table 4-5 lists a number of software measurement issues as viewed by developers and managers [Hall et al. 2001]. Software practitioners are often afraid the measurement data will be used against them and will take too much time to collect and analyse [Hoffman 2000]. They further express concern that software measures are too political and do not prove anything, or, that the team will focus on getting the numbers right rather than building good software [Wiegers 1999]. Productivity measurement, as the ratio of size over time is an example. Some developers might be tempted to write unnecessarily longer and inflated code to improve their productivity ratings. There is also an argument that, while using measurement practices might raise the rate of project success to a higher level statistically, this is only a valid issue at the organisation level, not at the individual project level. The reason is that projects usually have very short-term strategies and tight deadlines and, therefore, dislike sustaining certain costs in exchange for eventual organisational-wide gains [Meli 2000].

No	Software measurement issues	Developers %	Project Managers %	Senior Managers %
1	Hard to measure what you want to measure	15	25	0
2	Do not know how or if the data is being used	38	8	0
3	Detracts from the main engineering job	8	8	50
4	Difficult to collect, analyse, and use	23	58	50
5	Time consuming to collect data	38	67	25

Table 4-5 Negative aspects of software measurement [Hall et al. 2001]

By attempting to measure a software property, an assumption is made that the software property can be measured and that there exists a validated relationship between what is being measured and what is to be determined. In practice, however this is not often the case. Furthermore, while the evaluations of the external attributes are often the aim of the measurement, they cannot be directly measured. Only internal attributes can be directly measured. It is difficult to relate what can be measured through direct measurement to desirable external quality attributes. Often, mistakenly, a linear relationship between components of a measure is assumed [Erdogmus 2008a]. An example is the defect density metric, which is used to gauge software quality. If the defect density is calculated to be 1.7 per KLOC for a software size of 10 KLOC, we cannot assume that the defect density would be 17 KLOC for the software when the size is increased to 100KLOC.

A well-defined and consistent approach for assessment and review of development process activities is essential, which can be achieved through software process measurement [Fenton and Neil 1999b]. There are however difficulties in measuring process activities since they require active and concurrent assessment, rather than retrospective analysis, often possible with software products. There is also the lack of universal acceptance of methodological techniques for software development, forcing organisations to adapt measurement procedures to the methodology in use [Nance and Arthur 2002]. A further problem is the difficulty in measuring an attribute in isolation. Often an attribute to be measured is dependent or associated with other influential factors. For example, code review quality is dependent on the thoroughness of the person carrying out the review to some extent. Eliminating such influential factors to ensure accurate measurements is often difficult.

It appears that there is hardly any software attribute, which can be measured repeatedly, consistently, and accurately. For example, over the decades there have been many attempts to measure the expected size of software products through metrics, such as 'Lines of Code (LOC)' and 'Function Points'. There are however problems with both metrics in producing an accurate and reliable measure of software size. LOC presents a measure of size only, in terms of program length, ignoring other attributes such as complexity and functionality. LOC further fails to consider factors such as verbosity of the programmer, the programming language, and environmental complexities such as skills, pressure, tool support, and computing platform. Lack of standard measurement method and language dependence is amongst other difficulties with this method of software size measurement. However, it should be acknowledged that LOC is one the oldest and most popular and widely used software size measure [Sommerville 2007] and has the advantage of being easy to collect - no other measure is as well understood [Bassman 1995]. Furthermore, LOC tend to be more uniform and suffer less from instability, due to low values, compared to coarse-grained size measures, such as number of function points or use cases [Erdogmus 2008b]. While function-points do not suffer from many of the weaknesses of LOC, such as language dependency, they are difficult to compute and contain a large degree of subjectivity (e.g. dependent on estimator). It is also questionable whether they truly measure functionality [Fenton and Pfleeger 1997]. Function-points are most useful for data-processing systems that are rich in input/output operations and it is difficult to estimate function point counts for event driven systems, making them unsuitable for productivity measurement [Furey and Kitchenham 1997][Armour 2002]. The size metrics are used in determining many software attributes, such as defect density and productivity, as well as the cost and duration of the project. However, since there are issues with size metrics, the accuracy of any metric that is a derivative of size (e.g. productivity and defect density) is also undermined. The LOC size metric has been used in this study to determine defect density and productivity (see Chapter 7).

Software measurement is increasingly becoming an important factor for software organisations, toward the path to capability and maturity, partly because it is a requirement of many standards such as CMMI and SPICE. However, the existing measurement programmes are unable to deliver the required capability [Lawler and Kitchenham 2003]. The advice offered by some international standards (i.e. ISO), on the measurements of such attributes as productivity, has been shown to be unreliable [Kitchenham and Colin 2007]. For maturity to be achieved there is a need for benchmarking, which requires consistent measurement convention and definition [McGarry 2001]. Such consistency is however difficult to achieve even within a single company. In a benchmarking study, Heires [2001] was unable to analyse 63% of the projects because of incomplete or unobtainable core metrics and incomplete projects. Furthermore, many projects lacked the necessary correctness and validity to be included in the benchmarking database. It appears, therefore, that currently measurement programmes suffer from both invalid and missing data, which, causes delays and reduces results validity, as well a lack of metrics standards, which reduces data comparability.

The arguments put forward in this section have elaborated on some of the concerns, shortcomings, and flaws in the practice of software measurement. While there are many research works and international standards encouraging the software community and organisations to establish rigorous measurement programmes, the progress seems to be slow. While software measurement may therefore be currently immature and flawed, it serves a useful role in producing better software. Gilb [1988] supports this view by writing, "Anything that you need to quantify can be measured in some way that is superior to not measuring it at all". It should however be borne in mind that measurement results may be subject to the flaws in measurement discussed above and may therefore contain a large margin of error. For example, it is inadvisable to rely on the measurement results of a single study to make a generalised conclusion about the true value and nature of a software attribute. There is an argument that, since currently software cannot be measured properly, it should be abandoned until such time that our understanding of software has enhanced enough to enable its proper measurement [Zuse 1998]. Despite flaws and immaturity, software projects can still benefit from a sound measurement process and many software maturity standards (e.g. CMMI) include a measurement component. Software measurement needs to be continued earnestly, both theoretically and empirically, if not for the usefulness of the results that they currently produce, at least for their value in the advancement of our understanding of software itself and the ways that it can be measured. Any advancement in software measurement would benefit software organisations in better controlling and evaluating software activities and products, to produce higher quality software. Software engineering, in comparison to civil and mechanical engineering, is a relatively young discipline and measurement process and practice is essential in helping it move forward towards robustness, when we can accurately and quantifiably measure and evaluate software attributes and software quality. Measurement based research projects, such as this, can play a small part in helping to improve our understanding of the software measurement implications and gradually enhance our understanding of software engineering in general.

Measurement provides the bases to evaluate quantifiably the benefits of new concepts or technologies and, therefore, experimental investigations, such as this project, would be practically impossible to conduct without it. The software community should accept, acknowledge, and account for the fact that currently results of software experiments, where software measurement is involved, may have large margins of error; the more convoluted and complex the measurement, the larger the margins of error. In any measurement process one of the main objectives should be to minimise such error margins through detailed considerations of the measurement environment and validity. The software community should endeavour to develop a culture of measurement based software engineering to help move it forward towards a truly engineering based discipline, such as civil and mechanical engineering. There is an on-going argument within the software community about whether software engineering is a true engineering discipline. Some argue that while software development can be categorised as engineering in the future once its problems are resolved through maturity, currently it is not an engineering discipline [McConnell 1998]. Without a proper measurement baseline, the term 'software engineering' is rather inappropriate and misleading. Software development can only be a true engineering endeavour when its attributes can be defined and measured properly, accurately, repeatedly, and consistently.

In the above sections, the software measurement topic, related to this study's experimental research method, was discussed. In the following section, the background and literature to software experimentation is discussed and reviewed. The devised and conducted experimental research method is covered in detail in Chapter 5.

## 4.11 Experimentation in Software Engineering

Experimentation has long been regarded as the optimal way to test causal hypotheses [Singleton and Straits 1999]. While acknowledging the limitations of measurement, experiments, and human sensory perception, Albert Einstein once said that no science could advance without good experimentation and measurement. An experiment is a procedure for collecting scientific data in a systematic way, in order to maximise the chance of answering an hypothesis correctly (confirmatory research), or to provide material for the generation of new hypotheses (explanatory research) [Festing 2002]. Experiments are used, for instance, to contradict existing theories, to validate measurements or to evaluate the accuracy of models. They can help build a reliable base of knowledge and thus reduce uncertainty about theories, methods, and tools [Tichy 1998]. They can lead to new, useful, and unexpected insights and open new areas of investigation. Experimentation can be further used to evaluate new ideas or products, such as processes, tools, or development methodologies. In many cases the experiments provide, not only the best way of effectively evaluating an idea or product, but also the only way [Oman and Pfleeger 1997]. In this study, the experimental method is employed as an evaluation mechanism to assess the effectiveness of a concept (i.e. software patterns).

Controlled experiments, in particular, offer several important benefits. In a controlled experiment, the results obtained from an *experimental sample* are compared against a *control sample* that is practically identical to the experimental sample except for the variable whose effect is being tested. Controlled experiments can be used to conduct well-defined and focused studies, to scrutinise and measure specific variables and the relationships between them. They help in formulating hypotheses by enforcing the clear definition of the question being studied, resulting in studies with well-defined dependent and independent variables and well-defined hypotheses [Basili 2007]. Furthermore, results produced by controlled experiment have the potential of being statistically significant. The experimental research carried out in this study is a controlled experiment, which is fully discussed in Chapter 5.

There is an increasing understanding in the software engineering community that empirical studies are needed to develop or improve processes, methods and tools for software development and maintenance [Sjoberg 2005]. Software engineering, in comparison to other disciplines, is young and can certainly benefit from experimental methods of analysis. However, experimental research is difficult, mainly because any flaws in experiment design, data collection, and data analysis, run the risk of invalidating the achieved results and conclusions. The quality of knowledge obtained by experimental research is related to the quality of the data collected and the degree of rigour employed in analysing them.

### 4.11.1 Experimentation Framework

Basili et al. [1986] proposed a widely accepted and implemented experimental framework. The proposed framework has the advantage of dividing the experiment into a number of independent and well-defined sequential phases. As experimental projects are all different, the framework does not attempt to prescribe a particular technique of carrying out an experiment, but outlines the points and elements that need to be considered. It guides the experimenter through the experiment process from initiation to completion. The

framework was therefore employed in designing and conducting the experiment in this study. The framework consists of four phases: 1) Definition, 2) Planning, 3) Operation, and 4) Interpretation. The Definition phase, which is the first phase of the experimental process, contains six elements, as described in Table 4-6. This phase sets out the initial and important aspects of the experiment to be conducted. The phase ensures that the aims and objective of the experiment is clear and that issues surrounding the environmental aspects of the experiment are well and unambiguously understood. In this phase, elements such as experiments object, purpose, and scope are defined.

The Planning phase of the experimentation process concerns the three elements of design, criteria, and measurement as listed in Table 4-7. Each element involves a number of activities that need to be considered in planning the experiment. The Operation phase of the experimentation process includes the three elements of preparation, execution, and analysis. For the preparation element, a pilot study could be used to confirm the experimental scenarios, organise experimental factors or inoculate the subjects. The data is collected and validated during the execution of the experiment. For the analysis of data, a combination of qualitative and/or quantitative methods can be used. Finally, the Interpretation phase of the experimental process consists of the interpretation context, extrapolation, and impact elements. In this phase, the impact of the experiment in terms of replication and application is discussed.

Parts	Description
Motivation	To understand, improve, validate or assess the effect of a certain phenomenon
Object	The object of a study is the primary entity under examination (i.e. an end product, or a process model)
Purpose	This could be, for example, to evaluate the effectiveness of a testing process, to predict system development costs, or assess the reliability of a software product
Perspective	Perspectives of the interested parties: developer, modifier, maintainer, project manager, customer, user
Domain	This can be two types: 1) Individual programmer or programming teams, and 2) the programmes or projects
Scope	Single project, Multi-project, Replicated project

Table 4-6 Elements of the definition phase

Design	Criteria	Measurement
Experimental Design	Direct Reflection of Cost/Quality	Metric Definition
Incomplete Block	Cost	Goal-Question-Metric
Completely Randomised	Errors	Factor-Criteria-Metric
Randomised Block	Changes	Metric Validation
Fractional Factorial	Reliability	Data Collection
Multivariate analysis	Correctness	Objective Vs Subjective
Correlation	Indirect Reflection of Cost/Quality	Nominal/Classification
Factor Analysis	Data Coupling	Ordinal/Ranking
Regression	Information Visibility	Interval
Statistical Models	Programmer Comprehension	Ratio
Non-Parametric	Execution Coverage	Absolute
Sampling	Size	
	Complexity	

Table 4-7 Elements of the planning phase

The experimental design for this study is discussed in detail in Chapter 5. There are a number of issues that make software experimentation challenging. Difficulties in designing and conducting software experimentation have meant the publication of fewer experiments in software engineering than in other engineering disciplines. In the following section, some of the major issues in software experimentation are discussed.

## 4.12 Software Experimentation Issues

Experimental studies in software engineering is time consuming and difficult to design and conduct [Shull and Basili 2004]. The two most important components of software experimental research are the experiment design (e.g. control of extraneous variables) and a sound and valid measurement process. However, software experimentation design is often challenging, partly due to the human factors involved in software engineering, which make the control of variables difficult and imprecise. Difficulties in software measurement, discussed in the previous



section, are the other major factors that make high quality software experimentation difficult to achieve. In this section, some of the major problems and issues in software experimentation are discussed.

#### 4.12.1 Flaws in Experiment Design and Conduct

Flawless experimental research in software engineering is hard to achieve. Poor statistical design and small-scale experiments over too short a period are amongst problems outlined in the literature [Fenton and Pfleeger 1994]. There are a number of questions that should be asked about any empirical research to judge the quality of its results and conclusions. These include [ibid]:

1. Is it based on empirical evaluation and data rather than intuition advocacy?
2. Does it have a good experimental design?
3. Is it a toy situation or a real situation?
4. Are the experiments appropriate to achieving the goals of the experiment?
5. Was the experiment run for long enough to evaluate the true effect of the change in practice?

However, unfortunately the reality of software experimental research is that only a small percentage would fully comply with all the criteria stated above. Many studies [Tichy et al. 1995] [Sjoberg et al. 2005] report on the lack of quality in the published experiments in software engineering. While it is ideal to conduct experiments that have flawless experimental designs and measurement process, that are based on real situations, and that run over a long period, such experiments would be extremely difficult for many researchers or research organisations to conduct, partly due to the high costs and often unavailable funds.

Experiment design and the measurement method and process have been important and major activities of this research. It has also been an objective of the research to adhere to the experiment design and measurement principles and guidelines (e.g. outlined by Kitchenham et al. [2002] and Basili et al. [1986]) to design a sound experiment that does not suffer from the serious flaws. Proper experimental design (within the resource constraints) is crucially important for an experiment to produce results that are accurate and valid. However, very few empirical study designs are, or claim to be, flawless [Perry 2000]. In designing an experiment, one has to consider a number of experimental errors that could creep in the process, which might affect the experiment results and conclusions [Fenton and Pfleeger 1997]. These include errors of experimentation (e.g. invalid and flawed design, observation (e.g. invalid and inaccurate data), and measurement (e.g. flawed and invalid measures and measurement process). These errors have the potential of having a damaging influence in an experiment, leading to wrong and misleading conclusions. There have been many published experimental works that have produced questionable results and conclusions, due to inappropriate experimental design. For example, Shneiderman [1977] indicated, through an experiment, that pseudo code should replace structured flowcharts as a means of program and design documentation, which caused many authors to advise against the use of flowcharts. However, a subsequent experimental study [Scanlan 1989] showed that structured flowcharts are preferable to pseudo code for program documentation, exposing a number of experimental flaws in the Shneiderman's study, such as overlooking several key variables in his experimental design.

#### 4.12.2 Subjects in the Experiments

A common criticism of experiments in software engineering is that, in most studies, the subjects are students, making it difficult to generalise the results to apply to the professional development environment [Sjoberg et al. 2002]. Students are mostly used as subjects in software engineering experiments because they are more accessible and generally inexpensive. However, using professionals in experiments could have many advantages, such as higher skill and experience levels, better use of professional methods and tools, and better teamwork. On the other hand, it is often impractical to employ multiple teams in industrial settings for the sake of completing experiments, and developing the same product a number of times using different methods or approaches. This is something that can be achieved in a student environment.

Planning and execution of empirical studies in industrial settings are complex and expensive, because they may require a great deal of time, effort, and resource. The use of students in empirical studies provides a way of reducing technical and organisational risks and research costs. There are many situations where student subjects are either suitable or preferred. These include [Carver 2003]:

- Obtaining preliminary evidence to confirm or refute a concept, theory, or technology
- Controlling factors that may affect the study
- Showing software organisations the relevance of the research

- Provision of useful evidence to encourage software organisations to conduct further empirical studies
- Fine-tuning the organisation and details of an empirical study, before it is carried out in an industrial environment

In many experiments, the subjects are part-time students and will have had professional level work experience in software development industry [Sjoberg et al. 2002]. Many studies, such as this research, use student subjects that are very close to graduation and entry into a professional environment. Indeed, some might have already had some professional-level experience in software engineering, not least during their placement year. The significant difference between student and professional subjects is not, therefore, always clear-cut. A professional subject, who has just begun professional work, would be little different to a student subject who is very close to graduation and who may have already had several years of software engineering experience in a previous career. Furthermore, in some cases, students may be better suited to some experiments than inexperienced professionals. For the benefit of software experimental research, it would be helpful if software organisations planned and organised their developer's time, in such a way that they could allocate some of their time to participate in experimental research. It would also be helpful if the academic institutions, such as universities, also designed their courses in a way that participation of students in experimental studies would be easier and indeed encouraged.

### 4.12.3 Costs and Publishing Limitations

Experimental research is expensive to conduct and often requires more resources than non-empirical research [Sjoberg et al. 2002]. However, experimental research may often provide the only practical means of confirming or rejecting a theory or concept and, cost considerations, should not prohibit researchers from performing detailed and high quality experimental studies. Both private and public sponsors should view such costs, as long-term investment in software research and development.

Constraints on time and cost are the reason that a lot of research work is done in small groups of students, rather than in large-scale applications in commercial situations [Fenton and Pfleeger 1997]. However, it is generally acknowledged that small investigations are better than no investigation at all. Furthermore, small experimental projects may be appropriate for an initial venture into testing an idea, indicate directions for further investigation, or test a research design and generally improve understanding and raise new questions.

A further issue is that experimental studies are difficult to publish. Although experiments are conducted in the real world and are therefore always flawed in some way, experimenters often confront reviewers who expect perfection and absolute certainty [Tichy 1998]. In addition, many established journals seem to have difficulty in finding editors and reviewers, capable of evaluating experimental work [ibid]. In encouraging more empirical research and experiments in software engineering, it is important that the reviewers appreciate the inherent difficulties involved in such research and be more lenient in their criteria for accepting such research for publication.

### 4.12.4 Human Factors

There are some aspects of software engineering, compared to other science and engineering disciplines such as mechanical engineering and physics, which make software engineering experimentation more complex and error prone. One of these aspects is the intense human factors involved in software engineering. Most software is designed, constructed, tested, managed, and used by humans and when measuring something as abstract as software, human related factors (e.g. human characteristics, varying psychological and social aspects) come into play, that make accurate evaluations of many software attributes challenging and multidimensional. Software engineering is considered as a social process and, as such, is influenced by relationships among people involved in the social context (e.g. corporate culture, organisational procedures) [Juristo and Moreno 2001]. It is not often possible to accurately evaluate all the influencing human factors in an experiment a priori, in a deterministic manner, and need to rely on statistical methods to estimate their influence. There is an argument that, because software engineering is a social process, it is inappropriate to view it as a natural process, such as in physics with deterministic (rather than stochastic) effects to causes [Pfleeger 1999]. This implies that it requires a different method of study, one that is based on a stochastic rather than deterministic approach. Such stochastic approach to software experimentation would still need to follow the traditional methods of observing phenomena, formulating explanations and theories and testing them [Tichy 1998]. The direct involvement of humans with such complex psychology, cognition and social behaviour, is an aspect of software engineering experimentation and measurement that makes such endeavours more complicated and challenging. Such difficulties often deter researchers from doing experimentation in software engineering [Juristo and Moreno 2001]. In many

experimental designs however, such as the one designed in this research project, the extraneous effects of the human factors are minimal or neutralised, due to the random nature of the experimental groups and subjects (i.e. any differences are randomly spread between the experimental and control groups).

#### 4.12.5 Experiment Quality

A number of studies have surveyed the quality and quantity of the published software experiments. They report on a lack of quality in software experimentation in terms of experiment design and measurement. The studies indicate that in majority of cases the standard of experimentation quality, in terms of both the experimentation process and the analysis of the outcomes, is low. The quality of the experiments is weak, partly due to design flaws, lack of validation, and appropriate statistical methods to draw appropriate results and conclusions [Koziolek 2005].

Tichy et al. [1995] conducted a survey, which studied over 400 research articles and studied the experimental validation methods that they employed. Articles included those published by ACM Transactions on Computer Systems and IEEE Transactions on software engineering. There were also articles from other disciplines, such as *neural computing* (NC) and *optical engineering* (OE). These two areas were chosen for comparison purposes, because NE is relatively new (similar to Software engineering), and OE is, in contrast, an old and established discipline. The study indicated that, over 40% of computer science papers and 50% software engineering papers on design and modelling, completely excluded experimentation. However, only 14% of the NC and OE articles contained no experimental evaluation. Furthermore, computer science papers contained a significantly lower number of purely empirical studies than those in NC and OE. The articles with hypothesis testing were rare, at only 1% in all articles. While in NC and OE 67% of the papers dedicated 20% of their space to experimental validation, this proportion was much lower, at 31%, in computer science. The study therefore seems to disprove the common perception, which attributes the insufficient experimentation in computer science, to the relative young age of the discipline. At the time the study was undertaken, the NC discipline was only six years old, but contained an established level of experimentation comparable to a much older discipline, such as OE. It therefore appears that the relatively small number of experiments in software engineering compared to other disciplines may be largely due to the lack of well-established experimentation and measurement culture and techniques in software engineering. This is a factor which has been supported by some studies such as [Koziolek 2005].

A similar study carried out by Zelkowitz and Wallace [1998], in which 612 Software Engineering papers, published in IEEE Transactions on Software Engineering, IEEE Software, and the International Conference on Software Engineering (ICSE), and 137 papers from other disciplines (i.e. Physics, and Psychology), published in various corresponding journals, were reviewed and investigated. The results of this study were analogous to the findings from the study carried out by Tichy et al. [1995] in terms of the quality and quantity of software experiments. The study showed that almost a third of the articles studied had no experimental validation at all. Only 30% of the articles had limited experimental validation, in which the experimenter and the subjects were themselves developers of the products or technology under study, and included an inappropriate level of control. Such experimentation is often referred to as *pseudo experimentation*, whose results may be highly biased and therefore not reliable. The study further found that the experimentation goals and objectives were not defined explicitly, clearly, and unambiguously.

A further investigation of the quality of software experiments was carried out by Sjoberg et al. [2005] who surveyed over 5,400 scientific articles, published in leading journals and conferences from 19 countries. The study found that only 1.9 % of the work involved and performed controlled experiments. One reason, for such a small percentage of controlled experimentations, is due to the large resource necessary for conducting well-designed experiments [ibid]. The study showed that the number of subjects participating in the experiments ranged from four to 266, with a mean value of 49, with approximately 75 percent of the subjects being students. The study further showed that reports were often vague and unsystematic and that there was often a lack of consistent terminology. A strength of this study compared to others (for example Tichy et al. [1995], and Zelkowitz and Wallace [1998]) is in the large number of articles that were surveyed, as well as in the structure and the detailed content of the report. A weakness of this report, however, is that statistical significance analysis has not been performed and, therefore, it is uncertain whether the results provided are statistically significant. This may somewhat compromise the validity and accuracy of some of the results.

### 4.13 A Review of Pattern Related Experiments

There have been comparatively few published experimental studies on software patterns. While there are numerous patterns proposed in the literature, attempts at empirically validating such patterns or evaluating their

usefulness, are relatively few. Some of the reported experiments on evaluating patterns are reviewed in this section.

Prechelt [2002] carried out two similar experiments to assess the usefulness of design pattern documentation in program maintenance. Subjects performed maintenance tasks on two programs, ranging from 360 to 560 LOC, including comments. The experiments were designed to test whether it helped the maintainer if the design patterns in the program code were documented explicitly (using source code comments), compared to a well-commented program without explicit reference to design patterns. The subjects were a combination of undergraduate and graduate computer science students. The number of subjects for the first experiment was 74 (64 graduates, and 10 undergraduates). For the second experiment, there were 22 subjects, all of whom were undergraduate students. All the subjects received a few weeks of training on design patterns before the experiment. The subjects were divided into two groups of experimental and control groups, where the experimental group received source codes with design patterns explicitly commented (called Pattern Comment Lines - PCL) as some extra comments. The control groups however received the source codes where design patterns were not commented explicitly. The performance of subjects was investigated by assessing the completion time, grading answers, and counting correct solutions. The following two hypotheses were tested:

- Hypothesis H1:* By adding PCL, pattern-relevant maintenance tasks are completed faster.
- Hypothesis H2:* By adding PCL, fewer errors are committed in pattern-relevant maintenance tasks

The experiments confirmed both hypotheses, and therefore supported the explicit use of PLC for design patterns. The main strength of this work is in the design and conduct of the experiment, in a field with few previous experimental investigations. The work is also elaborative and detailed in terms of its discussion of the validity issues of the experiment. Amongst other strengths of the work is its comprehensive statistical analysis of the results, which includes an evaluation of the statistical significance of the results. However, the work suffers from a number of weaknesses, the main one of which is a weakness in the experiment design. The experimental groups were offered comments on the design patterns (i.e. PLC), in addition to the comments that both the experimental and control groups received. That means that the experimental groups had more lines of comments than the control groups. It would have been more appropriate if the general comments, regarding the design patterns, were replaced by the design pattern comments (PLC), rather than added to the general comments count. In addition, while the number of subjects for the first experiment is a reasonable number of 74, the number of subjects for the second experiment was low at only 22.

A further experiment to assess the effect of design patterns on the maintainability of software applications was carried out by Prechelt et al. [2001]. The aim was to test if design patterns should be used, even if the actual design problem is simpler than that proposed by the pattern (i.e. not all of the functionality offered by the pattern is actually required). The hypothesis to be tested was 'A design pattern, P, does not improve performance of subjects doing a maintenance exercise, X, on program, A, (containing P) when compared to subjects doing the same exercise, X, on an alternative program, A, (not containing P)'. The experiment used three independent variables (i.e. programs and change tasks, the program version, and the amount of pattern knowledge) and two dependent variables (i.e. time and correctness). A total of 29 (originally planned 32, but 3 did not participate) subjects, all professional software engineers with average professional programming (C++) experience of 2.4 years, were used. Fifteen of these subjects had already had some experience of design patterns. The subjects were divided into 4 groups (6 to 8 subjects per group), in which each group maintained one pattern program (containing design patterns), and one Alt program (not containing design patterns), with two or three work tasks for each. A number of GoF's design patterns [Gamma et al. 1995] (Observer, Visitor, Decorator, and Abstract Factory) were used, for which the subjects received two days of training. The groups were compared before and after the design patterns training (i.e. pre-test and post-test), having been asked to perform a number of maintenance tasks on four small software programs.

The results of the experiment indicated that the use of the Observer pattern, in a simple program, had a negative effect on maintainability and the Visitor pattern was neutral. The Decorator pattern had a positive effect, and the Abstract Factory pattern caused only small differences. Although the study did not indicate a clear positive effect of some of the design patterns in the context of the experiment, it can be argued that, unless there is a clear reason to prefer a simpler solution, it would be wise to use the design pattern solution for the flexibility that it would provide in handling possible future requirement changes. The experiment was well designed, and the fact that the experiment involved two phases (i.e. pre-test and post-test), made the argument for the validity of its

outcomes and results stronger. The results were also statistically analysed and presented. There were, however, some issues that may be considered as the weakness of the experiment. These are as follows:

- *Small number of subjects:* Although measures were taken to ensure that the groups were randomly selected and were similar in the relevant abilities, only four groups were involved.
- *Familiarisation:* A two-day design pattern course was probably too short for the subjects to fully understand the design patterns under examination.
- *Generalisation:* Only four design patterns were used in this experiment. The results are therefore applicable to the examined patterns and cannot be generalised to include all patterns.
- *Context:* The experiment did not take place in a programming environment and the subjects used pen and paper for their answers, rather than implementing and testing them in a real programming environment.

In an attempt to verify the results achieved by Prechelt [2001], Vokac et al. [2004b] replicated the experiment. In contrast to the original experiment, where 29 students were used as subjects, in this replication 44 paid professionals (39 professionals from 11 companies, and 5 PhD and MSc students) took part as subjects. The experiment also took place in a real programming environment, instead of being a pen and paper exercise. The data from the original experiment was reanalysed, using the same regression model and estimation method, to enable the comparisons between the results of the two experiments.

The results differed from the original experiment [Prechelt 2001], particularly in the case of Visitor and Observer design patterns. While the original experiment found the Visitor pattern to have a neutral effect on maintainability, the replicated study found that it had a negative effect. Furthermore, in contrast to the original finding, this experiment indicated that the Observer pattern did not have a significant negative effect. The general conclusion reached was that the tested design patterns had their own characteristics and, it was therefore not valid, to characterise such patterns as useful or harmful to the maintenance activities. While the two experiments somewhat contradict each other, the result of this replication may be more valid and reliable, as it has a number of advantages over the original experiment. These are: 1) A larger number of mostly professionals were used, and 2) There was improvement in the experiment environment. The environment included a non-intrusive logging software, to measure elapsed time, and saved, time-stamped copies of every file compiled. The data provided by the logging system resulted in a more extensive quantitative analysis. Furthermore, this replication conducted a more detailed statistical analysis of the results, than the original experiment.

An experiment to investigate the effect of design patterns on communication between developers was carried out by Unger and Tichy [2000]. This was done in order to test the claim that design patterns improved communication between the members of the development team [Buschmann et al. 1996]. The experiment compared two-person teams, with and without pattern knowledge, communicating about program designs. Verbal communication was captured with audio and video devices and the transcripts were analysed. Communication was considered more effective if there were to be clear episodes of explanations and balanced discussions during design work. The teams received a program design (containing design patterns) for maintenance and were required to discuss how to design a number of given requirements changes into the existing design. This took place in two phases (i.e. before and after the teams attended a three-month course on design patterns). The results indicated that team communication improved in the post-test, compared to the pre-test. These results therefore showed support for the claim that design patterns improved communication between software developers.

There were however some weaknesses in this experiment that could have had an effect on the validity of the experiment's conclusions. One of the weaknesses is the small number of subjects used. Although there were plans to use 7 teams (14 student subjects), some of the subjects did not participate in both pre-test and post-test phases of the experiment and only 5 teams (10 subjects) fully participated. There was also a three months interval between the pre-test and post-test phases of the experiment. Although during the three months the subjects attended a course on design patterns, it is possible that they could have gained skills and knowledge, other than design patterns, which caused their communication performance to be improved in the post-test phase.

Porter and Calder [2004] tested the applicability and usefulness of design patterns in teaching programming to novice programmers through an experimental research. In this experiment two groups of students were selected (experimental and control), where the experimental groups were given a set of design patterns to use for their assignment, while the control groups used non-pattern solutions. The works were evaluated upon the completion of the assignments using a five level scoring mechanism (excellent, very good, good, satisfactory, poor). The evaluation was based on the assessment of the quality of the works in terms of programming technique and style.

It is not however stated what patterns were used in this experiment. A weakness of this experiment was the use of a relatively small number of subjects (only 18). Furthermore, the work did not include any statistical significance analysis of the results. Although the results achieved proved to be inconclusive, it showed that researchers are, seriously considering the applicability of design patterns, as an aid in pedagogy.

The effectiveness of design patterns in generating better quality designs were studied by Reibing [2001\_b] through experimentation. He examined two sets of designs one of which used the State design pattern [Gamma et al. 1995], and the other used no design patterns. The two designs were then compared for quality. The results of this study proved to be interesting for the fact that two contradictory conclusions were achieved, depending on how quality was defined. Using conventional OO quality metrics (i.e. WMC, DIT, NOC, CBO, see Table 4-4), the study showed that contrary to the expectation, the metrics results indicated, that the designs that did not use design patterns were of better quality than those that did. The result leaves two interpretations: 1) the metrics that were utilised were not good indicators of design quality, and 2) the application of design patterns in software design reduces the quality of the resulting design. However, if flexibility is to be an indicator of quality, then the designs using design patterns proved to be of a higher quality. This exemplifies the subjective nature of software quality and the inherent difficulty in its definition and measurement. A software application could be considered to be of high quality in one definition and of low quality in another. Generally, there should be a more appropriate notion of software quality that incorporates complexity criteria such as size and coupling, and the flexibility considerations. A weakness of this work is that the results are based on a single and relatively small software program and no statistical analysis of the results were carried out. For the results to be fully valid, the experiment should have been conducted on a sufficient number of applications where the results could have been statistically validated. The results achieved cannot be, therefore, generalised.

#### 4.14 Summary

In this chapter, experimentation and measurement in software engineering, which are two main topics of this research, were discussed. As software engineering is relatively young in comparison to some other engineering disciplines (e.g. civil and mechanical), it has not reached a desired level of maturity with respect to both experimentation and measurement.

There are mainly two types of measurements in software engineering: direct and indirect. Direct measurement refers to the measurement of an attribute, when no other attribute has a direct or indirect influence (i.e. No. of lines of code). On the other hand, indirect measurements are used when an attribute can only be measured in relation to other attributes (i.e. efficiency, complexity, reliability). Indirect measurements are normally made using direct measurements. Both types were used in this research.

Software quality is difficult to define and has been the subject of much discussion within the software development community. While the term "quality" might seem self-explanatory, there are many different views of what is meant by software quality and how it should be measured or assessed. There are mainly two widely employed models of software quality measurement: a) Factor Criteria Metric model, and b) Goal Questions Metric model (GQM), which were discussed in the chapter.

The establishment of a measurement process in software development organisations is encouraged in the literature and by the international standards. In any measurement process, 'what is measured' and 'how a measurement is made', should be carefully planned and considered. Furthermore, the benefits gained by a measurement programme should be weighed against any disadvantages such a programme may cause. That is, care should be taken by the managers and measurement processes designers, not to include measures that may be unnecessary or damaging (e.g. assessing and comparing an individual programmer's productivity), to the morals of the organisation's workforce.

Software experimentation quality is currently low. There is a lack of validated and controlled experimental studies in software engineering due to many reasons, such as, the cost and difficulties in carrying out high quality experimentation. Controlled experimental research, such as the one designed and implemented in this research, contributes to and advances the scientific knowledgebase on experimentation in software engineering.

In the next chapter, the design and conduct of this study's experimental research method is discussed in detail.

---

## Chapter 5 Experimental Methodology

---

### 5.1 Introduction

In any comprehensive research programme, the research method plays a crucial role. Indeed, the validity of the research findings may depend on the suitability, appropriateness, and thoroughness of the applied research method [Christensen 2006]. It was therefore a major priority to devise a well designed an appropriate research method to produce valid results.

The research design for this study involved a *controlled experimental research method*. The background and a literature review of software experimentation in software engineering were presented in the previous chapter (Chapter 4). The experimental research method designed for this research was based on designs that are often associated with research in psychology, involving human subjects. Application of such research methods in software engineering is far less prevalent [Seaman 1999], and one of the contributions of this research programme is the implementation of such experimental research methods in the field of software engineering. In this controlled experiment, the experiment subjects were divided into *experimental (treatment)* and *control groups*. The experimental groups received the treatment in the form of process patterns to use in their software development projects. The control groups were not given access to the process patterns (i.e. the treatment). It was expected that the final analysis of the results would highlight a difference between the two groups, which could be contributed to the application of the treatment. to the experimental groups.

In the first section of this chapter, the experimental definitions and hypothesis are introduced. This is followed by an overview of the experiment and the issues involved. The experiment definitions and hypothesis is discussed next followed by a discussion of the experiment design. The process of conducting experiments and the ethical issues concerned are discussed towards the end of the chapter.

### 5.2 Experiment Definitions and Hypothesis

It is more than a decade now since the concept of software patterns was conceived. While there have been numerous papers and books on software patterns over the years, there have been few empirical studies of software patterns to evaluate their utility and value in software development. Furthermore, almost all of these studies have focussed on a single type of pattern, namely the 'design pattern'. In this experimental research, another type of software pattern (i.e. 'process patterns') is empirically studied to evaluate their utility and value in software development process. There has been a great deal of work in both scientific and industrial contexts towards identifying, writing up, and building support tools for software patterns. However, empirical studies on the effects of patterns are rather rare. While there have been some empirical studies to evaluate the effect and value of design patterns (patterns concerned with software architecture and coding [Gamma et al. 1995]) on various aspects of software development [Prechelt 2001, 2002], there appears to be no credible empirical studies to investigate the utility and value of process patterns. This study aims to address this issue by presenting an empirical study on the effect and value of process patterns.

The purpose of this study was to evaluate the utility and value of the application of process patterns on a software development project. The study, conducted through an experimental research method, assessed the effect of the application of process patterns on 260 software development projects. There were two types of projects (128 individual projects and 132 group projects) under investigation in this study, which were the results of two live university modules, involving software development projects (CMT3991, and CMT3992). The two project types were:

- **Group projects** (Module CMT3991, Computing Project Management). This module was a 12-week duration module (one semester), in which students worked in teams of 5 individuals, on a software development project to develop a software application.
- **Individual Projects** (Module CMT3992, Undergraduate Computing Project). This module was also a 12 weeks duration module, in which individual students (who passed CMT3991) worked on their own, distinct, software development project, with the help and advice of a supervisor.

The subjects for the experiment were final year undergraduate degree students who took modules CMT3991 and CMT3992 discussed above. The study took place at Middlesex University in London, involving three campuses where the two modules involved (i.e. experiment objects) ran.

The aim of the experiment was to investigate the following research question:

*How does the application of process patterns in the management of a software development project affect the quality of the project?*

Based on the research question, the null and alternative hypothesis to be tested was:

- $H_0$  Application of process patterns in the management of a software development project will *not* improve the quality of the project
- $H_1$  Application of process patterns in the management of a software development project will improve the quality of the project

While there are many proposed development approaches (e.g. waterfall, iterative ...) proposed in software development projects, they generally include four main phases or activities (i.e. Requirement analysis, Design, Implementation, and Delivery) in their development lifecycle. In investigating the research question, the effect of process patterns on each of these four main development phases is investigated through an experimental research method, which is discussed in this chapter. This involves the measurement and evaluation of a number of software project attributes through metrics in each of the four main phases of the development lifecycle. The metrics are selected through a measurement process discussed in Chapter 6.

### 5.3 An Overview of the Experiment Design

In this section, an overview of the experiment plan and design is presented. The following statements of facts are bullet pointed to describe concisely the circumstances of the experiment:

- The experiment was conducted across three campuses at Middlesex University, namely, Trent Park (TP), Tottenham (TM), and Hendon (HE).
- The experiment was conducted during two semesters.
- CMT3991 students at Trent Park (TP) campus were in the treated groups in semester one, and were in the control groups in semester two.
- CMT3991 students at Hendon (HE) were in the control groups in semester one (Sem1) and in the treated groups in semester two (Sem2).
- CMT3991 students at Tottenham (TM) were in control groups in semester one. The CMT3991 module did not run in Sem2 at TM, and therefore no student from TM took part in this semester.
- CMT3992 module was involved in semester two only
- CMT3992 students at Trent Park (TP) were in treated group, and those in Tottenham (TM) and Hendon (HE) were in the control groups

These statements are further illustrated by Table 5-1, Table 5-2 and Figure 5-1.

Phases	Semesters	Module	Campus	Status
One	One	CMT3991	Trent Park	Treated
			Tottenham	Control
			Hendon	Control
Two	Two	CMT3991	Trent Park	Control
			Hendon	Treated

Table 5-1 Experiment arrangements for the group projects

Semester	Module	Campus	Status
Two	CMT3992	Trent Park	Treated
		Tottenham	Control
		Hendon	Control

Table 5-2 Experiment arrangements for the individual projects



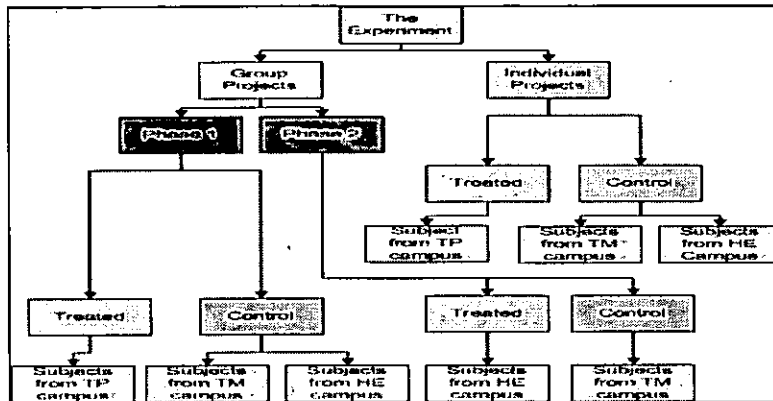


Figure 5-1 Experiment Design

The experiment was carried out in two phases (across two semesters), where the status of the treated and control groups alternated between the semesters and the campuses, as illustrated in Table 5-3. This is to ensure that any changes between the treated and control groups is independent of the status and specifics of the campuses and the semesters in which the experiment is conducted. Therefore, any variation between the treated and control groups can only be attributed to the application of process patterns and not variations or differences in the semesters or campuses.

<p><b>Phase 1</b> = Semester One  <b>Phase 2</b> = Semester Two                  ✓ = Treated                  × = Control  <b>TP Campus</b> = Subjects taking module CMT3991 at Trent Park Campus  <b>HE &amp; TM Campuses</b> = Subjects taking module CMT3991 at Hendon or Tottenham sites</p>		<b>Phase 1</b>	<b>Phase 2</b>
	<b>TP Campus</b>	✓	×
<b>HE &amp; TM Campuses</b>	×	✓	

Table 5-3 Experiment design

As well as data gathered through measurement process, the official marks offered to the projects by tutors were also considered in this experiment. Figure 5-2 depicts the structure of a section of the experiment, in terms of the two sets of data captured and analysed.

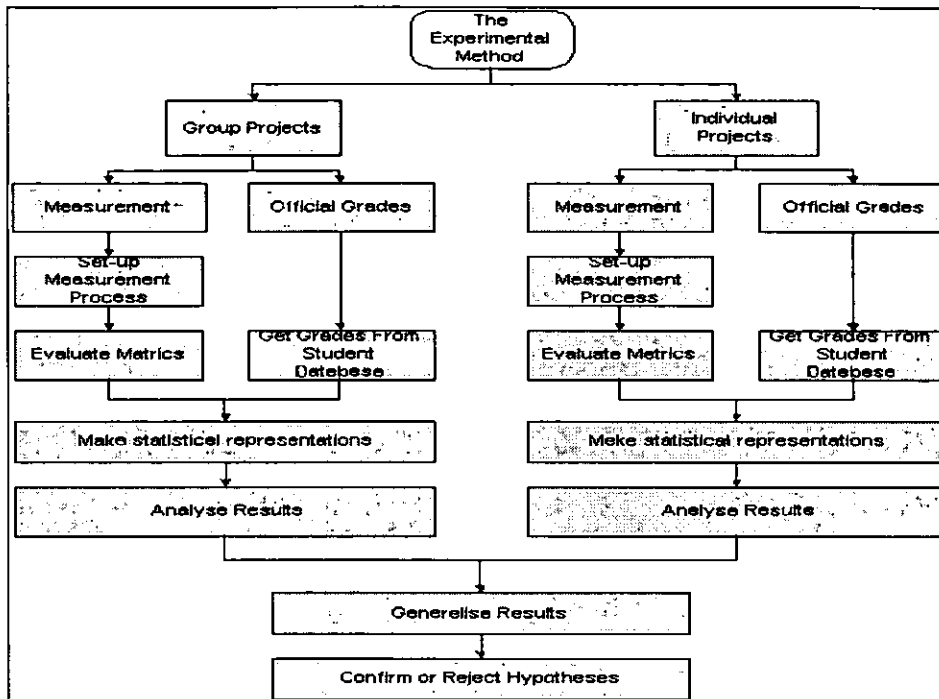


Figure 5-2 Capture and analysis of data to test the research hypothesis

A number of important issues had to be carefully considered in designing the experiment. In the following section, the issues concerned are outlined and discussed.

## 5.4 An Overview of Issues Involved

As this particular experimental study was the first of its kind to be carried out at the Middlesex University with no precedence, many issues had to be considered and resolved. These issues can be categorised into the following:

- Practical and logistical Issues
- Ethical concerns
- Staff concerns

These issues are discussed in this section.

### 5.4.1 Practical Difficulties

In designing the experiment, many issues and questions had to be considered and answered. These are listed as follows:

1. What type of experiment method would be appropriate for the study?
2. What are the variables involved?
3. What are the extraneous variables and what measures should be taken to control them?
4. How will the experimental (i.e. treated or conditioned) and control groups be selected? Will it be based on voluntary or compulsory participation of the subject?
5. What should be the sample size in order that the results could be analysed for statistical significance?
6. Should the subjects be told about their participation in the experiment?
7. How can the treated and control groups be matched?
8. Should subjects be selected from students on the same degree courses (programmes)?
9. Should the experiment be split across campuses?
10. How will the subjects receive the treatment condition?
11. How to make sure that the subjects use the given treatment (i.e. the process patterns)?
12. What incentives can be used to encourage subjects to use the treatment?

13. How to ensure that the treatment given to the experimental groups will not leak to the control groups?
14. How will the software projects be assessed in order to detect any differences between treated and control group as a result of the treatment condition?
15. What will be the tutor's influence on the outcome of the projects?
16. What would be the researchers influence on the outcome of the projects?
17. How could the effect of differences and discrepancies in tutors marking and abilities be minimised on the outcome of the experiment
18. Can the selected groups work on different projects or should they be given the same project title?
19. What are the ethical issues concerned with the experiment?
20. How to ensure that subjects are treated fairly and equally, irrespective of their roles in the experiment?

The issues listed and their respective resolutions in the design of the experiment are discussed in this chapter.

### 5.4.2 Ethical/Staff Concerns

One of the major issues to consider was the ethical issues involved in using students in the experiment. Many questions had to be answered and resolved in this regard, to satisfy Middlesex University's Ethics Committee. These ethical concerns are discussed in detail later in the chapter in Section 5.8.

The other issue was the concerns of lecturers and seminar tutors teaching the courses that were to be used in the experiment. Naturally, staff concerned with the courses, especially the teaching staff and the course leader, wanted to ensure that the whole experiment was carried out ethically and fairly, with little or no extra work and responsibilities for them. Fortunately, after many meetings with the staff concerned, and many iterations and modification of the experiment's design, their approval was achieved for the experiment to go ahead.

## 5.5 Experiment Specification

The design for this research programme includes a method of enquiry for its suitability and appropriateness for testing the research hypothesis (i.e. the research question). In this section the experiments research setting, experiment variables, control measures, and validity issues will be discussed.

### 5.5.1 Experimental Research Settings

There are different types of experimental approaches, which differ in terms of their applicability in different situations and settings. In designing the experiment for this study, the two main experimental approaches (i.e. a laboratory setting, and a field setting) were considered. A laboratory experiment is a study that is done in the laboratory in which the experimenter manipulates one or more variables and controls the influence of the extraneous (unwanted) variables. Laboratory experiments provide the best way to control or eliminate the influence of extraneous variables [Shaughnessy 2002]. This is accomplished by bringing the problem into an environment different from the subject's normal settings. Although in such environment outside influences could be eliminated, there is a price to pay in terms of the artificiality of the situation, which may not necessary reflect the real situation. A field experiment is an experimental research that is done in a real life setting. Here, the experimenter manipulates variables and controls the influence of as many extraneous variables as possible. In contrast to laboratory experimentation, field studies are not generally subject to the artificiality problem. Their primary disadvantage, however, is that the control of extraneous variables cannot be accomplished as well as with laboratory experiments [ibid].

The experimental method for this research programme is a field experiment, since it is to be conducted in a real life situation and setting. Although one can argue that student's projects are artificial because they do not deal with real life situation (e.g. business), they are real since they have targets and objectives that are apart from the experiment. In other words, the projects were not being done for the sake of the experiment. In that sense, therefore the study could be classified as a field study.

### 5.5.2 Variables

An experiment contains a number of different types of variables. A variable is some property of an event in the world that has been measured [McBurney 2003]. Variables in an experiment are entities which are subject to variation and whose values are observed by the researcher. One advantage of the experimental approach is that it provides excellent control techniques, allowing the researcher the ability to manipulate variables and observe

their effects [Christensen 2006]. The variables involved in this experiment will be discussed in the following sections.

### 5.5.2.1 Independent Variables

The independent variable is the variable whose value is changed by the researcher, within a defined range, and whose effect on the other variables is monitored and recorded. It is the variable that, according to the hypothesis, creates the presumed effect [Singleton and Straits 1999]. The desired variation in the independent variable can be achieved in various ways. The following are two options for the treatment variable considered for implementation in this research:

**Presence versus absence:** In this technique, one group of subjects receives the treatment condition and the other group does not. The two groups are then compared to determine if the group that received the treatment differed to the groups that did not, with respect to the dependent variables.

**Amount of variable:** In this technique, different amounts of the variable are administered to each of the several experimental groups. This technique can be used to find the minimum or maximum amount of treatment required to induce a difference between the groups.

The *presence versus absence* technique was used for its suitability and applicability in this research. The other option was rejected, as the experiment's objective was not to study the effect of an individual or a particular number of process patterns, but to determine whether the application and usage of process patterns, as a whole, would produce any effect. Furthermore, treating a large number of process patterns individually, as independent variables, would be unnecessary and impractical in terms of statistical analysis. There were two independent variables in this experiment, which were the treatment (presence or absence of process patterns) and the semesters (Semester 1, and Semester 2), in which the experiment was conducted.

### 5.5.2.2 Dependent Variables

The dependent variable is the variable that measures the influence of the independent variable. By changing the value or status of the independent variable (i.e. presence or absence of process patterns), it is presumed that there will be changes in the value of the dependent variables. By observing, monitoring and recording the values of these dependent variables, it will be possible to verify whether the research hypothesis is confirmed or rejected [McBurney 2003].

In this experiment, we were interested in the effect of process patterns on different phases of a complete development lifecycle, by measuring and evaluating a number of attributes from each phase. The artefacts (i.e. documentations, models, source code, test plans) produced in the four major phases of the development lifecycle, as well as the development process in each phase, are the components of the dependent variables. The dependent variables are the attributes to be evaluated for the four major phases of a development lifecycle (i.e. Requirement Analysis, Design, Implementation, and Delivery). The dependent variables are listed in the Table 5-4.

Attributes
Requirements traceability
Requirement specification reviews
Granularity of modules
Comment density
Source code review
Defect density
Productivity
Defects removal ratio (for each development phase)
Test time allocation (for each development phase)
Test case density

Table 5-4 The independent variables

In an ideal world, we would want the independent variables to be the only variables affecting the dependent variables. However, in the real world, there are often other variables that would also affect the dependent variables [Kitchenham et al. 2002]. These variables are referred as the *extraneous variables*. Extraneous

variables are the unwanted variables such as, intelligence, past experience, learning ability, programming skills, and the experimenter's effect that should be considered, planned for, and controlled, in order to minimise their influence on the dependent variables.

### 5.5.3 The Treatment

The experiment's treatment was a set of process patterns used by the experimental groups. The following strategies were considered in selecting and preparing the set of process patterns:

- 1) Selection of a small number of individual and specific process patterns
- 2) Selection of a complete system of process patterns covering a complete development lifecycle

While option one initially appeared to be preferred for its specificity and simplicity, it suffered from the following disadvantages:

- Process patterns are generally linked and related to each other, and it would therefore be impractical to isolate individual process patterns and evaluate their effect on specific software quality attributes.
- It limits the scope of the study to specific process patterns rather than a complete system of patterns. Any results would therefore apply to those specific patterns, rather than a set of process patterns covering a complete development lifecycle.

It was therefore decided to adopt and implement option 2, to study the effect of a complete system of process patterns covering the whole development lifecycle (i.e. Requirement analysis, Design, Implementation, and Delivery). Therefore, in compiling the list of process patterns to be used, there were two primary objectives. Firstly, the selected list of process patterns had to cover a complete development lifecycle, and secondly, the compiled process patterns had to be appropriate for the type of development projects under investigation.

A number of sources of process patterns were investigated for their suitability for inclusion in this experiment. Amongst the sources were a two volume-book on process patterns [Ambler 1998, 1999] that covered a full development lifecycle, and a set of process patterns proposed in an influential paper by Coplien [1995] (see Section 2.5). There were also a number of other sources from which process patterns were extracted to be used in the experiment which included [D'souza and Wills 1999] and [Storrie 2000] (see Section 2.5). While some of these patterns were suitable enough to be included in the compiled list as they came in their source, most had to be edited to reduce their size to present a succinct version of the patterns. A total of 98 process patterns were compiled for the purpose of the experiment. Some of the process patterns used in the experiment are presented in the Appendix B. Patterns.

The evaluation method was designed to measure the overall affect of a system of process patterns on software development projects through metrics, which measured a number of software attributes. The objective of the experiment was not to determine whether the employment of any particular process pattern had an effect on the quality of a software attribute, but to gauge the collective influence of the whole system of process patterns. There is a many-to-many relationship between the presented process patterns and the attributes to be measured. That is, one or more process patterns could affect a single attribute and the related metrics. Accordingly, a single process pattern could affect one or more software attributes and the corresponding metrics. This relationship is generally dependent on the type of the process pattern and its position in the process pattern hierarchy. The higher-level process patterns in the hierarchy (i.e. more generic) would have a wider scope and would therefore affect a higher number of attributes and the related metrics.

For example, 'define and validate requirement' process pattern (Appendix B. Patterns) would influence any attribute (and the associated metrics) related to the definition and validation of requirements (e.g. requirement ambiguity rates, requirement review quality). The pattern, 'developing in pairs' could have an effect on many attributes and corresponding metrics across all the phases of the project (e.g. productivity, defect density, etc). Therefore, single process patterns could affect multiple attributes of the development project and in turn affect the value of the metrics employed to measure them. Similarly, a single software attribute and the metric measuring it could also be influenced by one or more process patterns. For example, 'defect density' may be affected by many process patterns of different hierarchical levels (e.g. code ownership, review of architecture, etc). An attribute and the related metrics could be affected by all the individual patterns in the complete set of process patterns. This many-to-many relationship between process patterns and the software attributes is depicted in Figure 5-3.

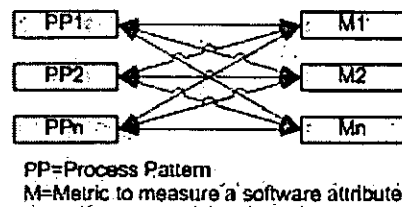


Figure 5-3 Many-to-many relationship between process patterns and metrics

### 5.5.4 Control

Extraneous variables cannot be eliminated and thus it is not possible to eliminate their influence from the experiment. It is however possible to eliminate any differential influence that these variables may have across the various levels of the independent variable [Campbell and Stanley 1963]. In other words, it is possible to keep the influence of these variables constant across the various levels of the independent variable.

For this experiment, a number of extraneous variables were identified and ways of controlling them were devised. For example, the experimenter in this study played a part in subjecting the experimental groups to the treatment. Experimenter effects have been defined as the unintentional biasing effect that the experimenter can have on the results of an experiment. Experimenter is not just a passive, non-influential agent in an experiment but could in some cases be an active and potential source of bias [Rosenthal 1998] [McCrone 2004]. The experimenter could also have a positive effect in reducing bias. They can help standardise the extent to which all subjects understand the instructions [Aronson and Carlsmith 1968]. Furthermore, the experimenter may be necessary to detect the occurrence of unanticipated phenomena, that could affect the outcome of the experiment, and to identify ways of improving the experiment. In the final analysis, the possible gains of having an experimenter must be weighed against the possible bias that they may introduce.

In this experiment, all efforts were made to ensure that the experimenter's effect would not undermine the validity of the experiment. The experimenter was responsible for introducing the process patterns to the experimental groups through lectures and seminars. The experimenter was aware that anything beyond an introduction to process patterns, to inform the experimental groups on how to access and use process patterns, should not be offered to the subjects.

There is an established principle stating that the act of observing and conducting an experiment could affect the outcome of the experiment [Landsberger 1958] [Parsons 1974]. This is referred to as the Hawthorne Effect and states that an individual's behaviour may change if they are aware of being studied. This was based on an experiment that showed that the productivity of employees seemed to improve, regardless of the employed experimental manipulation. In other words, experiment subjects' performance could improve as a result of just being participants in the experiment, referred to as Hawthorne Effect. This effect was considered and influenced the final design of the experiment as discussed in the Section 5.6.

Errors resulting from the misunderstanding of data can be minimised, if the person recording the data remains aware of the necessity of making careful observations and ensures that data are accurately recorded [Sjoberg et al. 2002]. Another approach would be to use multiple observers or data recorders. In order to reduce errors, the data recording was done online through a single portal. This means that data was entered once only to the system by the participants. The data was then made available to be accessed by statistical analysis packages or other tools for analysis.

Experiments can use a number of techniques in selecting subjects in order to minimise the effect of extraneous variables. Two such techniques are referred to as randomisation and 'matching'. Randomisation is a statistical control technique that has the purpose of providing assurance that known, or unknown, extraneous variables will not systematically bias the results of the study. It is one of the main techniques to control the known sources of variations [Shaughnessy 2002]. Random selection of subjects provides assurance that the sample is representative of the population from which it is drawn, and therefore ensures that the extraneous variables are controlled. In order to implement a randomisation technique one, ideally, should randomly select subjects from a potential pool of subjects. These subjects should then be randomly assigned to the same number of groups as there are treatment conditions. The treatment conditions should then be assigned to the experimental groups. This is illustrated in Figure 5-4.

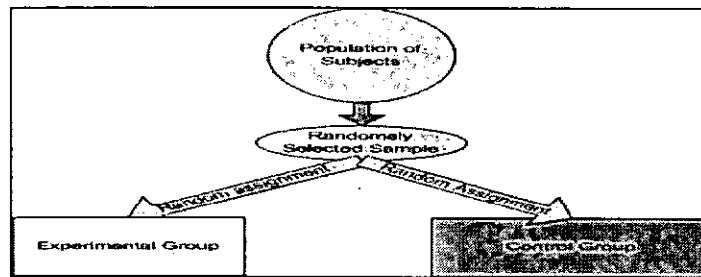


Figure 5-4 Random subject selection

Although randomisation provides the best guard against interpreting differences in the dependent variables, as being the result of variables other than the independent variable, it is not however the best technique for determining the sensitivity of the experiment [Christensen 2006]. The sensitivity of an experiment can be increased by matching the subject to the various experimental groups. A second benefit of matching is that the variables, on which subjects are matched, are controlled as constancy is achieved. For example, if subjects in all treatment conditions are matched on intelligence, then the intelligence level of the subjects is held constant and is therefore controlled for all groups.

A matching technique for controlling the extraneous variables, and increasing the sensitivity of the experiment, is to equate subjects on the variable or variables to be controlled. If intelligence, for example, needs to be controlled, then subjects in each of the experimental groups are at the same intelligence level. The technique of precision control [Sellitz 1959] requires the experimenter to match subjects in the various experimental groups, on a case-by-case basis for each of the selected extraneous variables. The matching technique is illustrated in Figure 5-5

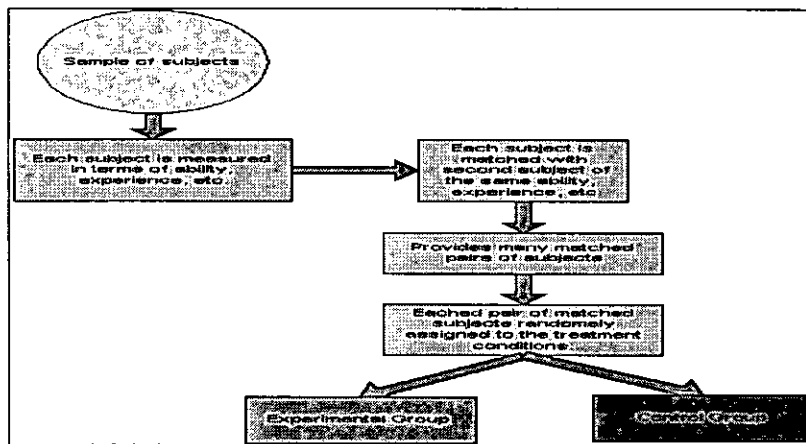


Figure 5-5 Matching by precision control technique

Both randomisation and matching techniques were considered for use in this experiment. After considering advantages and disadvantages of each technique, as applied to this particular experiment, it was decided that, due to the availability of relatively large sample size, the randomisation technique would be the most appropriate technique to be used for this experiment. It was therefore decided that the matching technique would unnecessarily complicate the experiment, without providing a significant advantage over the randomisation technique.

Despite all efforts to control the effect of extraneous variables in the experiment, full control cannot be practically achieved. Christensen [2006] writes, "It can never be for certain that complete control has been affected in the experiment. All that can be done is to increase probability that we have attained the desired control of the extraneous variables that would represent sources of rival hypothesis".

### 5.5.5 Internal Validity

Internal validity describes the extent to which research design and process affect the results in terms of the effect of extraneous variables and is the most important and widely considered validity type McBurney [2003]. Experiments are internally valid when the obtained effect can be unambiguously attributed to the manipulation of the independent variable [Kitchenham et al. 2002]. In this experiment, it was important that the measurement data the students were to provide was valid. For this, the students had to be honest in the data they provided as well as diligent to ensure that the data was correct. It was made clear to the students by the tutors and the researcher that the measurement data they provided had no influence on the official marks they were awarded for their project. They were requested therefore to be as honest as possible in providing the actual data without any exaggerations. There was a risk that the subjects' answers on the measurement form might be influenced by either what they thought should be a good answer to the measurement questions, or by what they thought the officials would like to see. It was important to take steps in minimising such risks. It was therefore made clear to the subjects that they had to try to be truthful in providing measurement data for the experiment, and that the best data were those that truly reflected the reality of the situation in their development project.

It was made clear to the subjects that they should carefully and seriously consider each of the measurement questions and answer them carefully. The importance of carefully considered answers, in reaching correct results and conclusions in this research, was explained to them. The importance of their input to the research, in helping to make a possible contribution to the advancement of our knowledge of software engineering, was further explained to them. It was therefore important that they were honest and diligent in providing measurement data in this research

Random assignment of the subjects to the treatment and control groups ensures that any deficiencies in honesty and diligence (discussed above) are uniformly spread in both treatment and control groups. Therefore, for the purpose of this experiment, which involves comparing the performances of the two treated and control groups, such factors do not affect the validity of the experiment objectives in any significant way and, therefore, would not undermine the experiment's internal validity.

A number of measurements were required to be taken by the researcher. These were generally measurement of the quality of some software attributes. An evaluation method was devised through which the attributes were methodically and consistently evaluated. The other important aspect was to ensure that there was no bias in the evaluation of the projects, by the researcher. It is sometimes likely for a researcher to become biased towards a concept, often unwittingly, due to one's belief and views and, therefore undermine the integrity of an experimental study. It is therefore important that the researchers are aware of this fact and take extra care to ensure that their input to the project does not suffer from bias [McCrone 2004]. Such cautious approach, in the evaluation of projects by the researcher, was taken to eliminate any bias that could be damaging to the validity of the results.

The respective universities lecturers marked the completed projects. In marking the projects, a number of aspects of the development project were evaluated and marked. Although they were not evaluating all the aspect of the development, in the way they were evaluated by the researcher, their evaluation marks provides another set of measurement data, which were used in the experiment, to compare the treated and control groups. As the treated and control groups were marked by the same tutors, any difference in marking is randomly spread between the treated and control groups, and therefore had negligible effect the internal validity.

### 5.5.6 External Validity

External validity concerns the generalisation of the experiment to other situations. As the circumstances and environment in which the experiment could take place might differ considerably, there are many external validity issues. The following are the three main issues covered by external validity [Christensen 2006]:

- **Other subjects:** Would the experiment produce the same results with different subjects? (For example, if professionals were to be used instead of students)
- **Other Times:** Would the experiment produce the same results if it were conducted at another time?
- **Other setting:** Would the experiment produce the same results if it were conducted in other settings? (i.e. being done in industry on commercial projects, using different tools, and being under different pressure levels)

Work environment in industry is significantly different to universities. In industry, professionals are usually assigned to work on single development project, fulltime, with strict guidelines, demands, rules and constraints



issued by both managers and customers. The developed application should work perfectly in a real business environment. This is quite different to a university setting where students know that the application is not going to be used and, are therefore, much more relaxed.

It can be argued that if in a university environment, where students are under fewer constraints, a difference between the treated and control groups is noticed in the experiment, any difference might be more prominent in an industrial setting, where work is carried out in a more strict and disciplined manner, resulting in better application of the treatment. That is, for example, if the application of the process patterns showed an improvement in performance in a more relaxed environment, it is likely that a more stringent and focused application of process patterns, in a more strict and better controlled environment (e.g. commercial organisations), would show higher improvement levels. Furthermore, although students were used as subjects, rather than professional practitioners, they were in their third (final) year of studies, close to their professional start in industry. Therefore, it is perhaps possible to make a cautious assumption that, if the experiment was done using professional subjects, the experiment would have produced similar results. However, this is a hypothesis that needs to be tested in a replication and could be the subject of a future work.

Having discussed the fundamental concepts and various aspects of control issues within the experiment, in the following section, design of the experiment is discussed.

### 5.6 Experiment Design

An overview of the experiment design was presented in Section 5.3. The experiment design is broadly discussed in this section. The design issues such as the involvement of subjects, the method of applying treatment, and the method of selecting subjects, as well as the design models are discussed. Figure 5-6 depicts the overall structure of the experiment design.

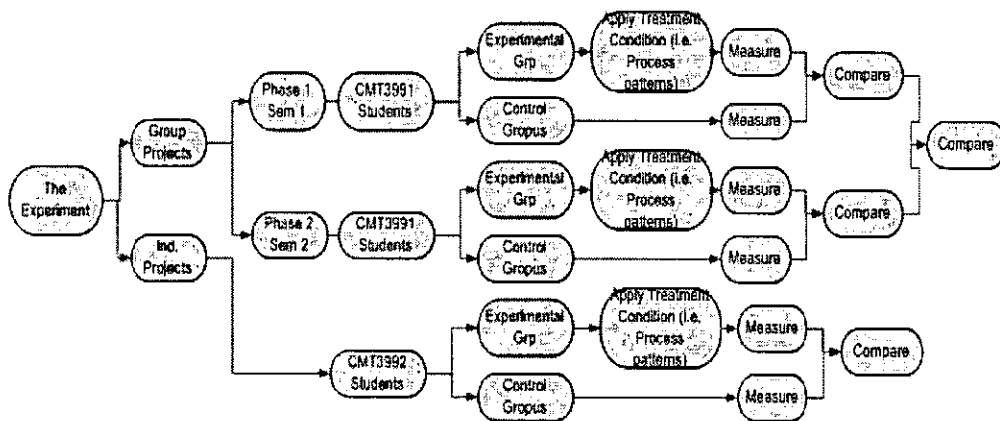


Figure 5-6 Experiment design

#### 5.6.1 Design models

As depicted in Figure 5-6, while in the case of group projects the experiment was conducted in two phases (semesters), it was conducted for a single semester in the case of individual projects. The design model devised for project types therefore differed accordingly. While the number of dependent variables was the same across both project types, the number of independent variables varied. In the case of group projects where the experiment was conducted for two semesters, there were two independent variables, the treatment (presence or absence) and the semesters (Sem1 and Sem2). By treating the ‘semester’ as independent variable, the effect of changes in semesters can be analysed to verify that any difference between the treated and control groups are not due to changes in semesters (i.e. Sem1 and Sem2), but as a result of the application of the treatment.

For the group projects, therefore, the experiment design was modelled on an approach referred to as ‘2 × 2 between subjects factorial design’, also referred to as “two-way ANOVA design”, where there are two independent variables involved. The design is referred to as “between subjects design” because the same subjects were not used in both phases of the experiment (i.e. Students doing group projects in semester 1, were different to those in semester 2). The experiment design in terms of its independent variables is depicted in Table 5-5.

Independent Variable A (Treatment)	Independent Variable B (Semesters)	
	Semester 1	Semester 2
Treated (presence of treatment)	Treated groups in Semester one (Trent Park campus)	Treated groups in Semester two (Hendon campus)
Control (absence of treatment)	Control groups in Semester One (Hendon & Tottenham campus)	Control Groups in Semester two (Trent Park campus)

Table 5-5 The 2 × 2 experiment design (independent variables)

In the case of individual projects, where a single semester is involved, there was only one independent variable, which is the presence or absence of the treatment. The experiment design for the individual projects therefore used the 'independent samples t-test' for their analysis. Section 7.2.3 will further discuss the two devised experiment design models.

### 5.6.2 Subject's Awareness of the Experiment

In determining how much the subjects should know about the experiment, a number of options were considered. One option was to inform the subjects that the modules they were taking were being studied in an experimental research and, therefore, they were the subjects in the experiment. They would be informed of the details of their role in the experiment, such as whether they were in the treatment or control groups. This option ran the risk of the subjects in the experimental groups to perform to expectation, as they could assume that they were expected to, or should, perform better. This option therefore would have the considerable disadvantage of suffering from the Hawthorne Effect.

A second option considered was to inform the subjects of their participation in the experiment, without stating whether they were in the control or experimental groups. This option had also the disadvantage of suffering from the Hawthorne Effect, but to a lesser extent (i.e. subject's behaviour could be influenced by knowing that they are being participants in the experiment).

A third and final option considered was to decline to inform the subjects of their participation in the experiment. The experiment, therefore, would go on without the knowledge of the subjects involved. In this way, subjects would get on with the work as normal, without knowing that they were participating in an experimental study. This option had the benefit of sustaining complete normality in the way the subjects carry out their project work. While this strategy appeared to be the best, in terms of the integrity and clinical application of the experiment, it suffered from some disadvantages. Firstly, the implementation of this strategy would be rather impractical, as it would be difficult to keep the experiment a secret. More importantly, it would be ethically inappropriate that people should take part in an experiment without their knowledge and consent [Singer and Vinson 2002]. Therefore, the second option was chosen in which the students were informed of their participation in the experiment, without being told whether they were in the treated or control groups.

### 5.6.3 Subjects and Treatment Application

There had to be a way of ensuring that the subjects in the experimental groups would use the treatment condition. In other words, there had to be an incentive for the subjects to fully participate and use the treatment (i.e. process patterns). A number of possible solutions were considered. An option was to tell the subjects that the treatment would improve their work. This option did not however appear to have enough persuasion power to convince all the subjects to use the treatment.

The other option considered was to arrange for marks to be allocated for the use of process patterns (i.e. the treatment). In order to ensure that the control groups would also be entitled to the extra marks, they would also be set a separate task, for which they would receive a mark equivalent to that of the experimental groups. That, however, would mean modifying the current marking system for the module (group project) by adding other elements entitled for marking. The university used a computerised marking system called PAM, which did not allow modification of the marking criteria. That option therefore did not appear feasible for practical reasons. The marking approach is also detailed in documentation that was validated against the module. Changes to that documentation could require a partial re-validation of the module.

The final option was to assign the project coursework, so that using the treatment would be a requirement. In this design, the project assignments would be set such, that it would be specifically required of the experimental groups to use process patterns (i.e. the treatment) in their project. Adherence to this requirement is reflected in a marking criterion 'reflecting on their approach' in the PAM marking system. This option appeared to be the best option, as it did not suffer from the disadvantage that the other options entailed. This was therefore the option chosen to be used for this experiment.

## 5.6.4 Subjects Selection Methods

A number of possibilities for selecting subjects were considered and analysed. The following is a description of those that were considered, and the reason they were selected or rejected.

### 5.6.4.1 Voluntary-Based Selection

In this design, students were to be given the option of participating in the experiment. Once volunteered, they would then be assigned to either treatment or control group. In consulting relevant tutors and course leaders, it was decided that there was a high possibility that a sufficient number of students would not volunteer for the experiment. The problem was compounded by the fact the students for module CMT3991 module worked in teams of five and, therefore, all the members of the team had to volunteer to participate in the experiment. That made the likelihood of finding a sufficient number of teams, volunteering for the experiment, small. Furthermore, students that would volunteer to participate in the experiment might be from different seminars. This would be impractical, as the subjects could only receive help and instructions to use the treatment through their seminar sessions and, having a mixture of control and experimental groups in a seminar, would make the task impractical. This design was therefore considered impractical and was dismissed.

### 5.6.4.2 Mandatory-Based Selection

In this method the students from all, or a selected number of seminars, would be required to participate in the experiment. There would be two possibilities: 1) selecting a specific number of seminars for participation, and 2) selecting all seminars.

In order to minimise the effect for the seminar tutors on the experiment, seminars with same tutors for treated and control groups were required. Therefore, any tutor teaching a treated group would also need to teach a control group. Therefore, from the module timetable, those seminars could be selected which satisfied this condition. In selecting the seminars on this basis, there needed to be enough seminars shared between tutors, so that the above condition could be satisfied (i.e. each tutor taught one treatment seminar and a matching control seminar). There did not appear to be enough seminars shared between tutors from which seminars for the experiment could be selected (that is, not enough seminars so that each tutor could be assigned to a treated and a control group).

The other option of selecting all the seminars to participate in the experiment was considered. This option was deemed appropriate as it ensured that there would be a maximum number of subjects for the experiment and was therefore selected. There also seemed to be a concern regarding the leakage of treatment condition across the control groups, where there were both control and experimental groups in the same campuses. In order to prevent, or significantly decrease, the likelihood of control groups accessing the treatment condition, it was decided to choose the treatment and control groups to be from different campuses. In this design, all the seminars in a single campus would act as experimental groups for one semester and as the control groups in the next, and vice versa.

Students taking the module CMT3991 (group project) had a lecture and seminar-based tuition format. In each seminar, there would normally be four teams, and each team would normally have five members. The seminars were used to subject the experimental groups to the treatment condition. Therefore, all teams in the treated seminar groups received the treatment. Students that completed module CMT3991 (group project) in the first semester went on to take the module CMT3992 (individual projects) in the second semester. The students that took on a software development project were used as subjects. They were assigned to either experimental or control groups in accordance with their group status (treated or control) in the previous CMT3991 module.

Figure 5-7 below illustrates the way students in Module CMT3991 (group projects) are assigned to the experimental groups.

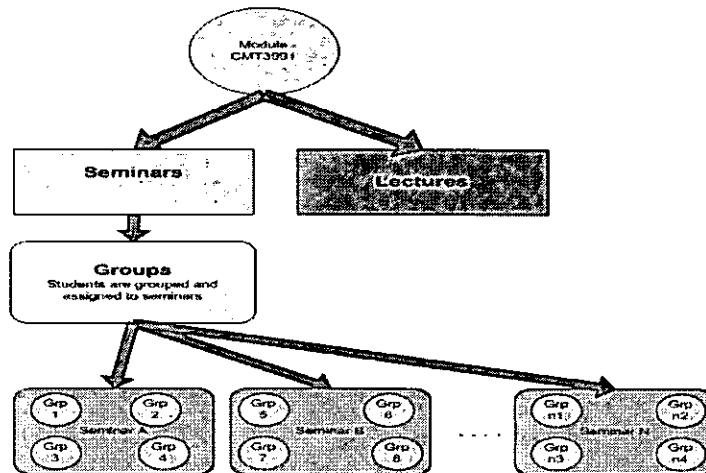


Figure 5-7 Module CMT3991 (group projects) seminar structure

In the following section, a number of issues about the details of the experiment design are discussed.

### 5.6.5 Group and Individual Projects Assignments

The project assignments for the module CMT3991 involved in the experiment had to be such that the work involved would test the subjects on all the elements of interest in the experiment. Group projects for CMT3991 were normally based on software application development. As such, any project title would be suitable for the experiment, as long as the subjects were told that they needed to follow a proper development lifecycle approach and document each phase. It was however decided that setting a common project title for all subjects to work on, would have the benefit of producing projects that would be easier to assess and compare. Therefore, in cooperation with the module leader and seminar tutors, the researcher devised a single project assignment for both treatment and control groups.

Students that completed and passed the CMT3991 module went on to take the CMT3992 module, which involved carrying out an individual project. The CMT3992 students chose the topic and title of their individual projects. Those projects, which involved the development of a software application, were selected for participation and analysis in this research.

Theoretical as well as practical aspects of the experiment design were presented in the above sections. In the following section, issues involved in the conduct of the experiment are discussed.

## 5.7 Experiment Conduct

In this section, the two main aspects of conducting the experiment are discussed. These are:

1. Application of treatment condition
2. Data types

Application of treatment condition is discussed first followed by a discussion of the data collection procedure.

### 5.7.1 Application of Treatment

Once the experimental and control groups were selected, there needed to be a mechanism for subjecting the experimental groups to the treatment condition (i.e. process patterns). The process patterns in the form of web pages were hosted on a specific website to which the treatment groups were given access. The treatment groups were also given introductory training on the process patterns and on how to access and use the materials on the website. The researcher for this experiment used both the lecture sessions, as well as the seminars to introduce the subjects to process patterns and answer questions about their use.

It was decided that rather than giving the subjects hard copies of the materials to use, it would be preferable to place them on a website for the following reasons:

1. **Provision of a facility to ensure the right subjects used the process pattern materials:** By issuing log on IDs and security measures it was possible to ensure that the materials were available to the treatment groups only
2. **Provision of facilities to gauge and determine levels of material usage:** By keeping a log of the subjects who logged onto the system, it was possible to determine whether the subjects were actually making use of the treatment. By keeping track of the data gathered in terms of who logged into the system, it was possible to ensure that all of the treatment groups were using the materials and reminded anyone who was not accessing the process patterns on the website to do so. Furthermore, the data collected would be used to determine if there was a correlation between, the number of times subjects logged in to the website to use the patterns and their performance in terms of the quality of the software attributes measured.
3. **Reduction in data errors:** Measurement forms were also hosted on the website, which were available to both treated and control groups. The online hosting of the measurement helped reduce data recording errors and facilitated data analysis. In addition, it had the benefit of the resources associated with the experiment to be available in one place.

Using an authentication system of username and password, the website presented a number of process patterns to the authorised subjects (i.e. treatment groups). As well as the actual process patterns, the website offered guidelines on how to use the materials. A sample of the process patterns hosted on this site is included in the 'Appendix B. Patterns'. The website also provided the facility for online forms for data collection and recording through measurement forms. The forms were available to both treated and control groups. The data collection procedure is discussed in the Section 6.3.1. The screenshots of the forms can be seen in 'Appendix A. Experiment Details'.

## 5.7.2 Data Types

Data for this experiment came from two main sources:

- Data from student records as marked by tutors
- Data collected from the measurement process

At the end of each semester, projects were submitted to the university to be assessed and marked by lecturers. A software application called PAM assisted tutors in marking and recording the student marks for the two modules (i.e. CMT3991, and CMT3992). The marking scheme used in assessing the project attributes was based on 5-point scaling system (excellent, good, average, poor, very poor). The details of the marking criteria and scheme are presented in the 'Appendix A. Experiment Details'.

A number of distinct project attributes were assessed and marked by the tutors individually. While most of these attributes were concerned with the actual project report (i.e. abstract, introduction, conclusion...), there were some that were directly related to the development efforts which were of interest in this study. These are:

1. **Design and analysis** (assessing the quality of design and analysis of the application developed)
2. **Product** ( assessing the quality of the end product)
3. **Evaluation** (assessing the quality of evaluation/testing methods applied)
4. **Project Management** ( assessing the management quality of the project )

The attributes marked are depicted in Table 5-6 in relation to their respective development phase.

Officially Marked Attributes	Development Phases
Design and Analysis	Requirement Analysis, Design
Evaluation	Delivery
Product	Delivery, Implementation
Project Management	Requirement Analysis, Design, Implementation, Delivery

Table 5-6 Relationships between the development phases and the marked attributes

In addition to the tutor marks, measurements of many software attributes spanning a complete development lifecycle were collected through a devised measurement process. The measurement process is discussed in detail

in Chapter 6. Once the projects were completed and submitted, they were officially marked by the seminar tutors as normal. The projects were then passed on to the researcher for evaluation. The quality of their content was evaluated by the researcher with regards to the attributes of interest. The subjects also submitted further measurement data through the online forms. Online measurement forms were used in the experiment as discussed in the Section 6.3.1.

### 5.7.3 Subjects' Views on Process Patterns

The treated group's views on their experience of using process patterns were sought through a questionnaire hosted online, which contained two 4-point Likert scale questions. The questionnaire contained the following questions:

1. How useful did you find process patterns in doing your project?

Not at all  Slightly  Moderately  Very

2. How difficult/easy did you find the process patterns to understand?

Very difficult  Difficult  Easy  Very easy

There are a number of possible outcome scenarios for the final analysis of the data. In the following section, some of these are discussed.

### 5.7.4 Experiment Outcome Scenarios

There were a number of possible outcomes with regard to the effect of patterns on performance. In this section, some of these are discussed and illustrated through graphs. Performance here refers to the value of the software attributes being measured in the experiment. As often customary in graphical representation of the experiment data of this type, the performance trends in all the following outcome scenarios are represented by lines, in order to make the relationship between the treated and control more graphically clear. It should therefore be noted that these are not lines in the sense of regressing a number of points on the graph.

One scenario is that the treated groups in both semesters would do better than control groups. Figure 5-8 shows performance levels for both control and experimental groups across the two semesters. It indicates that the control group's performance is constant across the two semesters and is lower in value than that of experimental groups, which is placed at a higher level. It also shows that the experimental group's performance is constant across the two semesters and is higher in value than that of control groups, which is placed at a lower level.

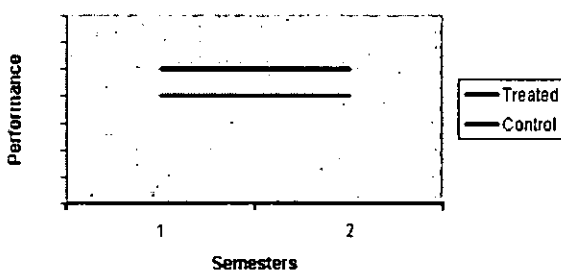


Figure 5-8 An outcome Scenario

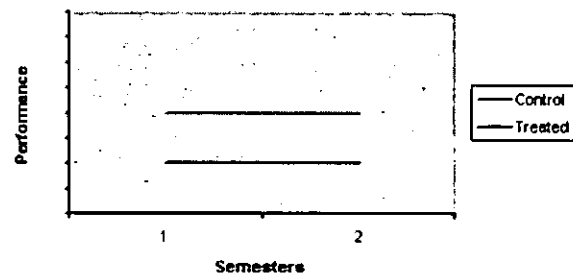


Figure 5-9 An outcome Scenario

Another outcome scenario is illustrated in Figure 5-9. The figure shows performance levels for both control and experimental groups across the two semesters. It indicates that the control group's performance is constant across the two semesters and is higher in value than that of experimental groups, which are placed at a lower level. The figure also shows that the treated group's performance is constant across the two semesters and is lower in value than that of the control groups, which are placed at a lower level.

The Figure 5-10 depicts a scenario in which the control group's performance is not constant across the two semesters. While the control group's performance is low in semester one, it is higher in semester 2. Similarly, the treated control group's performance is high in the first semester and low in the second. The Figure 5-11 depicts a further outcome scenario.

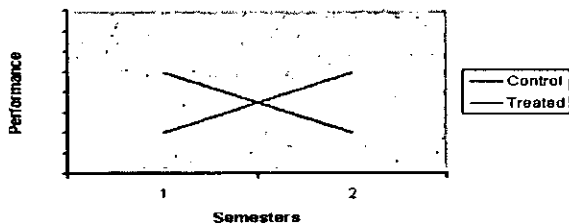


Figure 5-10 An outcome scenario

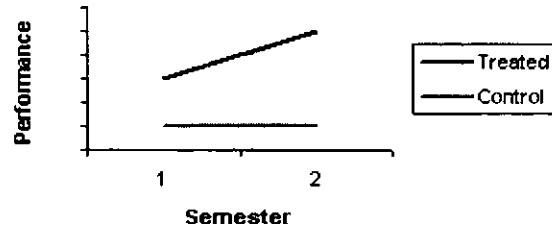


Figure 5-11 An outcome scenario

Experiments involving human subjects, must take all precautions to ensure that the experiment is wholly ethical and that no human subject is adversely affected by the experiment in any way. The ethical issues concerned with this experiment are discussed in the following section.

## 5.8 Ethical Issues

As the experiment for this research project involved human subjects, one of the main considerations was the ethical concerns involved. There were two key objectives involved here; firstly, identifying the concerned ethical issues, and secondly, addressing and resolving any identified issues [Duquenoy 2005\_a]. In order to ensure that scientific research is carried out in an ethical manner, an attempt has been made by some organisations to identify a number of ethical concerns and propose guidelines for their resolutions. The American Psychological Association [APA 2002] has produced a set of principles, referred to as the APA Code of Ethics, which aims to provide a set of ethical guidelines to researchers and research organisations that adopt it (See Appendix A. Experiment Details). However, adopting such a code of ethics could limit the field, scope and types of research and therefore before adopting the code, one must consider the consequence of adopting such code ethics [Shamoo 2002]. The API Code of Ethics were considered in this study and where appropriate adopted to achieve the two key objectives stated above.

Ethical concerns can be broadly identified in the following three different areas [Diener and Grandell 1978], which were considered in this project:

1. The relationships between society and science
2. Professional issues
3. Treatment of subjects

This experiment did not generally have any social effects and therefore produced no ethical concerns in relation to the first point. The professional issue is one that concerns all scientific research, and this research programme was no exception. It was a priority that the research programme was carried out objectively, accurately, and honestly.

The treatment of the experiment subjects was the most fundamental ethical issue in this study and was therefore given careful consideration and planning. In particular, the privacy and confidentiality issues of the subjects [Singer and Vinson 2002] were carefully considered. As a result, the experiment was designed and conducted in such a way, that no information about individual subjects, such as marks, or performance would be dispatched, published or divulged, which was outside and beyond the normal university practice, without the full agreement of individual subjects. The subjects' consent to allow their projects to be used in the experiment, were acquired through a consent form that they signed and submitted with their projects. In capturing and presenting the data and the analysis, careful consideration was given to aggregating and anonymising the data, so that none of it could be traced to its origin in order to identify a particular individual subject or subjects. It was a policy of this research to ensure that the experiment subjects did not suffer from any embarrassment as a result of their participation. As final year students, many of the subjects might have already been under stress from the pressure

of work. The experiment took this into consideration in planning the design and conduct of the experiment not to cause the experiment subjects undue stress.

In real life experimentation, one has to understand and appreciate the circumstances of the situation and the constraints and compromises involved. This is discussed next.

## 5.9 Design Constraints

In studying real life situations, and designing and conducting experiments within them, to learn and understand some phenomenon, or test some hypothesis, the researcher has to understand and take account of the environmental constraints and limitations involved. Therefore, the nature of this study, as real life experimentation, is such that it invariably brings constraints on the experiment design. Some of these issues are discussed below:

- Selection of subjects of the same abilities to both control and treatment groups
- Control of the amount of treatment condition given to the treatment groups
- Time given to accomplish project tasks
- Variation in team sizes
- Ethical issues

The key point is that the experiment design took all of those points into consideration and attempted to devise the best possible solutions in a way that the internal/external validity of the experiment was not adversely affected. These are explained in this section.

Students working on group projects had different abilities and characteristics. According to the university regulations, they had to form a group or a team to work on a project of their own choosing. The teams could not therefore, be specifically set up according to some criteria (such as their ability), for the purpose of this experiment. While the use of such a matching method might have been beneficial, it would have been generally difficult to judge accurately students abilities according to some criteria, and then match them. Because of the random nature of the experimental and control groups and the large number of subjects (sample size), any differences and discrepancies in the groups were randomly dispersed between the control and treatment groups. Any discrepancies would therefore be constant and would not adversely affect the results of the experiment.

It was not possible to control and measure the amount of treatment condition (i.e. process patterns) that the subjects used accurately. They were told to use as many as they needed for their project. However, while the number of times each subject accessed the process pattern pages on the website were recorded as a measure of the usage rates (discussed in Section 5.7.1), the system did not record which patterns were accessed and used by the subject. It would have been advantageous in terms of knowing the access rates of the used patterns, had the system recorded such data.

Although there was a set amount of time the subjects had to work on their project as dictated by the academic semesters, the actual amount of time they spent on development was based on their own estimation. They declared how much time they spent on development activities on an online measurement form. Subjects were instructed to fill in the forms as accurately and as honestly as possible. Their estimation of the time they spent in each phase had to be accepted as the actual time spent.

Although there was a recommended team size of five for group projects, according to the university's rules, the size of groups could change and in some cases did. However, this did not affect the experiment, as the project efforts were based on person-hours spent on the project. Furthermore, any change in the team size affected both treated and control groups, and was therefore a constant factor.

There is always an element of ethical concern in experiments involving human subjects, which have to be fully considered in the design and conduct of an experiment. Ethical issues concerned with this experiment had to be dealt with head on and from the first principles meeting requirements, such as fairness, confidentiality, and others. The experiment had to be devised in a way that would satisfy the University's Ethics Committee that all ethical issues were fully considered to prevent a breach of ethics.



---

## 5.10 Summary

In this chapter, the experimental research method with respect to its design and implementation, was presented and discussed. The purpose of the experimentation was to, evaluate and assess the utility and value of the application of process patterns, on a software development project.

The experiment involved two types of software development projects (group, and individual) and had a two-semester duration. In the case of group projects, the subjects were divided into teams and worked on a common software development project. In the case of individual projects, subjects worked on their own software development project.

A '2 × 2 between subjects factorial design', also referred to as 'two-way ANOVA design' was devised where the treated groups receive a set of process patterns to use in their development project. The control groups did not use the patterns. The aim was to determine if there were any difference between the treated and control groups, which could be attributed to the use of process patterns. The experiment's independent variables were the presence/absence of 'process patterns' and the semesters in which the experiment was conducted. The dependent variables were a number of attributes across the four major development phases (i.e. requirement Analysis, Design, Implementation, and Delivery). The experiment required a comprehensive measurement process for data collection and analysis. The data gathered through the measurement process, as well as the official tutor marks awarded to the projects, were used in the experiment. Both the treatment (i.e. process patterns) and the measurement forms were hosted online using a specifically developed website.

Many ethical and technical challenges had to be overcome in designing and conducting a sound controlled experiment. These issues were discussed in this chapter in detail. The measurement process devised for this will be discussed in the next chapter (Chapter 6).

---

## Chapter 6 Measurement Process

---

### 6.1 Introduction

In the previous chapter, the experimental research method was discussed, where the measurement process was one of its components. In this chapter the measurement process, devised and implemented to define and collect the required measurements for the experiment, is discussed in detail. A devised measurement process was necessary for the experiment to define the measurement goals which needed to be achieved, the measurements and metrics that were required to achieve the goals, and the means by which the required measurements could be acquired.

The focus of discussion in this chapter, is the adaptation of the Goal/Question/Metric (GQM) model [Basili and Weiss 1984], to create a tailor-made measurement process, appropriate and applicable to this study. The aim of the measurement process was to provide the programme and the mechanism to measure a number of software attributes, as part of the experiment design, in order to evaluate the effect of process patterns on software development projects. The measurement concepts and the GQM model, which underlies the measurement process, were discussed in Chapter 4. In this chapter, the three elements of the GQM, in relation to the objectives of the measurement processes, are defined and presented in detail. The full details of the goals, questions, and metrics involved are presented in a number of specific tables. There is also a detailed definition and specification of each defined metric, in table formats. However, due to the detailed and extensive nature of these tables, only a few of them are depicted in the body of this chapter and the complete set of tables is presented in the 'Appendix C. Metrics Specifications'. Other components of the measurement process, such as data collection procedure and the tools used, are discussed towards the end of the chapter.

### 6.2 Measurement Process Design

Software measurement process is often employed for the estimation of future products, evaluation and analysis of artefacts, structuring of the software process, improvement of techniques and methodologies, and the control of software process [Ebert et al. 2005]. In this project, the software measurement process was employed in the context of evaluation, to test the experiment's hypothesis that the application of process patterns in software development, improved the quality of the software development project. Designing and conducting a measurement programme is often a difficult endeavour and involve many intricate issues that have to be carefully handled [Briand 1997]. The measurement process procedures outlined in the literature, such as [Briand and Basili 1999] and [Fenton and Pfleeger 1991], as well as guidelines proposed by Practical Software Management [PSM], [NASA], and [ISO/IEC 15939] standards, were considered in drawing up the measurement process for this study. A great deal of attention was given to the design and conduct of the measurement process, to ensure that it was appropriate and that the process was properly and accurately implemented.

The goal-oriented measurement process designed and implemented was based on the GQM model, which was discussed in Section 4.8.2. This involved a number of steps and tasks as follows:

1. Define goals and sub-goals
2. Define questions to achieve goals
3. Define metrics to answer the questions
4. Define data collection procedure

Defining goals is the first element of the GQM model. Practice has shown the importance of specifying a measurement goal precisely, since the selection and definition of suitable and useful measures and models depends strongly on the clarity of these early decisions [Basili and Rombach 1988]. The goal in the GQM model has a number of elements that are depicted in a structure to assist the creation of clear and unambiguous goal statements as shown in Table 6-1.

Elements	Definition
Object of Study	What will be analysed
Purpose	Why will the object be analysed
Quality Focus	What property/attribute of the object will be analysed
Viewpoint	Who uses the data collected
Context	In which environment

Table 6-1 Goal Elements of the GQM model

The goal statement had the following format, where the blanks were filled in accordance with the requirement of any specific goal:

Analyse \_\_\_\_ for the purpose of \_\_\_\_ with respect to \_\_\_\_ from the viewpoint of \_\_\_\_ in the context of \_\_\_\_.

The goals in the measurement process were depicted in the format stated above. The measurement was based on the evaluation of some attributes in the four major development phases (Requirement Analysis, Design, Implementation, and Delivery) of the development lifecycle and the measurement goals were defined accordingly. As the 'process patterns' used in the experiment affected activities in a complete development lifecycle, there were numerous possible goals that could be set. However due to the scope limitations of this research, only a limited number of goals were set and analysed. There were a number of attributes in each development phase that could be measured, which directly or indirectly related to software quality. Three categories of the process and product attributes were selected that were appropriate for the measurement of the two project types (individual and group projects) under investigation:

1. **Artefacts:** Artefacts (such as code and documents) produced during each development phase
2. **Tests/reviews:** The testing/reviewing quality of each development phase
3. **Efforts:** Proportion of time allocated to each phase

The *artefact goals* were concerned with the physical artefacts produced in each development phase and generally involved product metrics. The goal was defined for the purpose of the evaluation of artefacts that were produced in any of the four major development phases. Each development phase produced a number of specific artefacts, which were measured for evaluation. The goal defined for each development phase therefore corresponded to the artefacts produced in that phase. The *test and review goals* were defined with the aim of evaluating the quality of tests and reviews, by assessing the implemented test and review process and product for each development phase. This, for example, involved attributes such as document and code reviews. The *efforts goal*, aimed to evaluate the proportion of time that was allocated to activities in each development phase. In particular, the goal was to measure the proportion of the development phase time that was spent in testing.

For each goal, there were one or more related questions that aimed to determine the measures and metrics, which were required to achieve it. There were many questions that could be set for any one stated goal. However, due to the scope limitations of this research in terms of time resource, a limited number of questions were selected for each goal (one or two questions per goal). These questions were presented in a number of tables (Table 6-2 to Table 6-13).

Metrics were the third and final element of the GQM model and provided the answers to the questions related to the measurement goals. Each question, within a goal, required one or more metrics for their resolution. There were many metrics (in isolation or in composition), that could have been used to answer any goal-related questions. However, due to limitations on the scope of this research, only a small number of metrics were used and analysed (one metric per question). The metrics, required to answer the measurement questions, were defined for all the major development phases discussed above, and are presented in the GQM tables in the following section.

### 6.2.1 GQM Tables

Having discussed the individual components of the GQM model for the measurement process, a number of GQM tables for each development phase were created. The tables contain three main sections corresponding to, the goal, question, and metric components of the GQM model. Detailed rationales for the metrics, stated in GQM tables, are given in metric specification tables in the 'Appendix C. Metrics Specifications'. Examples of the metric specification tables are presented and discussed in the Section 6.2.2. The measurement data for the

metrics were collected by the subjects and the researcher, through an online measurement form as discussed in the Section 6.3.1.

The QM tables for the Requirement Analysis (RA) phase are presented in the following tables (Table 6-2, Table 6-3, and Table 6-4).

Goal	
<i>Analyse</i> the requirement artefacts/documents <i>For the Purpose of evaluation</i> <i>with Respect to RA artefacts quality</i> <i>From the Viewpoint of the developer</i> <i>in the Context of group and individual projects</i>	
Questions	Metrics
What percentage of the requirements is traceable?	Percentage of traceable requirements (Traceable Requirements per Total Requirements Ratio)

Table 6-2 GQM for artefacts in the Requirement Analysis (RA) phase

Goal	
<i>Analyse</i> the requirement artefacts <i>For the Purpose of evaluation</i> <i>with Respect to the RA test/review quality</i> <i>From the Viewpoint of the developer</i> <i>in the Context of group and individual projects</i>	
Questions	Metrics
What percentage of requirements specification document is reviewed	Percentage of reviewed requirements specification
What proportion of the defects is corrected?	Percentage of defects fixed (Defect Fixed per Defects Detected Ratio)

Table 6-3 GQM for test and review in the RA phase

Goal	
<i>Analyse</i> the requirement artefacts/document <i>For the Purpose of evaluation</i> <i>with Respect to the RA effort</i> <i>From the Viewpoint of the developer</i> <i>in the Context of group and individual projects</i>	
Questions	Metrics
What percentage of the RA phase time is spent in testing?	Percentage of Phase Time Spent in Testing (Test Time Per Phase time ratio)

Table 6-4 GQM for effort in the RA phase

The QM tables for the Design phase are presented in the following tables (Table 6-5, Table 6-6, and Table 6-7).

Goal	
<i>Analyse</i> the design artefacts/document <i>For the Purpose of evaluation</i> <i>with Respect to the design artefact quality</i> <i>From the Viewpoint of the developer</i> <i>in the Context of group and individual projects</i>	
Questions	Metrics
What is the average class size	Number of methods per class (Methods per Class Ratio)

Table 6-5 GQM for artefacts in the Design phase

Goal	
<i>Analyse</i> the design artefacts/document <i>For the Purpose of Evaluation</i> <i>with Respect to the design</i> test quality <i>From the Viewpoint of</i> the developer <i>in the Context of</i> group and individual projects	
Question	Metric
What percentage of the design document is reviewed	Percentage of reviewed design document
What proportion of the defects is corrected?	Percentage of defects fixed (Defect Fixed per Defects Detected Ratio)

Table 6-6 GQM for test and review in the Design phase

Goal	
<i>Analyse</i> the design artefacts/documents <i>For the Purpose of Evaluation</i> <i>with Respect to the design</i> effort <i>From the Viewpoint of</i> the developer <i>in the Context of</i> group and individual projects	
Question	Metric
What percentage of the Design phase time is spent in testing?	Percentage of Phase Time Spent in Testing (Test Time per Phase Time Ratio)

Table 6-7 GQM for effort in the Design phase

The QM tables for Implementation phase are presented in the following tables (Table 6-8, Table 6-9, and Table 6-10)

Goal	
<i>Analyse</i> the Implementation artefacts/documents <i>For the Purpose of evaluation</i> <i>with Respect to the Implementation</i> artefacts quality <i>From the Viewpoint of</i> the developer <i>in the Context of</i> group and individual projects	
Question	Metric
What percentage of the lines of code is commented?	Comment Density (Comments per LOC Ratio)
What is the rate of defects per lines of code?	Defect Density
What is the productivity in the Implementation phase	Implementation Productivity (LOC over Implementation phase time)
What is the overall productivity	Overall Productivity (LOC over Total Project Time)

Table 6-8 GQM for artefacts in the Implementation phase

Goal	
<i>Analyse</i> the Implementation artefacts/documents <i>For the Purpose of evaluation</i> <i>with Respect to the Implementation</i> test/review quality <i>From the Viewpoint of</i> the developer <i>in the Context of</i> group and individual projects	
Question	Metric
What percentage of the code is reviewed?	Percentage of source code reviewed
What proportion of defects is corrected?	Percentage of defects fixed (Defect Fixed per Defects Detected Ratio)

Table 6-9 GQM for test/review in the Implementation phase

Goal	
<i>Analyse the Implementation artefacts/documents</i> <i>For the Purpose of evaluation</i> <i>with Respect to the Implementation effort</i> <i>From the Viewpoint of the developer</i> <i>in the Context of group and individual projects</i>	
Question	Metric
What percentage of the Implementation phase time is spent in testing?	Percentage of Phase Time Spent in Testing (Test Time per Phase time ratio)

Table 6-10 GQM for effort in the Implementation phase

The QM tables for the Delivery phase are presented in the following tables (Table 6-11, Table 6-12, and Table 6-13)

Goal	
<i>Analyse the Delivery artefacts/document</i> <i>For the Purpose of evaluation</i> <i>with Respect to the Delivery artefacts quality</i> <i>From the Viewpoint of the developer</i> <i>in the Context of group and individual projects</i>	
Question	Metric
How many test cases are defined per requirement	Test case density (Test Case per Requirement Ratio)

Table 6-11 GQM for artefacts in the Delivery phase

Goal	
<i>Analyse the Delivery artefacts/document</i> <i>For the Purpose of evaluation</i> <i>with Respect to the Delivery test quality</i> <i>From the Viewpoint of the developer</i> <i>in the Context of group and individual projects</i>	
Question	Metric
What proportion of detected defects is corrected?	Percentage of defects fixed

Table 6-12 GQM for test/reviews in the Delivery phase

Goal	
<i>Analyse the Delivery artefacts/document</i> <i>For the Purpose of evaluation</i> <i>with Respect to the Delivery effort</i> <i>From the Viewpoint of the developer</i> <i>in the Context of group and individual projects</i>	
Question	Metric
What percentage of the Delivery phase time is spent in testing?	Percentage of Phase Time Spent in Testing (Test Time Per Phase time ratio)

Table 6-13 GQM for effort in the Delivery phase

## 6.2.2 Metric Specifications

The metrics for answering the questions related to the specific goals were defined in the tables above. For each metric that is defined, a specification table is defined that fully elaborates it. The specification of the measures required in working out each metric are also presented in separate tables. The Table 6-14 shows the specification table for the first metric above. The measures required for working out the metric are shown in Table 6-15 and Table 6-16. The complete set of metric specification tables for all the defined metrics and their required measures are given in the Appendix C. Metrics Specifications.

<i>Metric</i>	
<b>Percentage of Traceable Requirements (Requirements Traced per Requirements Defined)</b>	
<i>Description</i>	
<b>Definition</b>	Measures the percentage of the requirements that are traceable (Traceable Requirements per Total Requirements Ratio)
<b>GQM Goal</b>	Requirement Artefact Quality
<b>GQM Question</b>	What percentage of requirements is traceable?
<b>Type</b>	Quantitative
<b>Evaluation Method</b>	As per formula below
<b>Applicable Phase</b>	Requirement Analysis
<b>Rationale</b>	Requirements traceability refers to the ability to describe and follow the life of a requirement, in both a forwards and backwards direction [Ramesh and Jarke 2001]. A requirement should be linked to a higher level document (i.e. source), which could be a higher-level system requirement, as well as downward to the design elements, source code, and test cases that are constructed to implement and verify the requirement [Davis 1993][Hull et al. 2005].
<b>Purpose</b>	This metric was used in the experiment to determine if there was any difference between the treated and control groups in terms of the percentage of traceable requirements. The metric will show whether as a result of using process patterns in the development projects, the treated groups will have a higher percentage of their requirements traced to design, test, and implementation.
<b>Scope</b>	Small software development projects
<b>Attribute to Measure</b>	Traceability of Requirements
<b>Metric Scale</b>	Interval
<b>Related Measure(s)</b>	Number of Requirements ( Measure 2), Number of Traceable Requirements (Measure 1)
	<i>Required Measurements</i>
Number of Traceable Requirements (NTR)	$\frac{NTR}{NR} \times 100$
Number of Requirements (NR)	

Table 6-14 Percentage of traceable requirements metric (Metric 1)

<b>Number of Traceable Requirements</b>	
<b>Definition</b>	Measures number of requirements that are traceable
<b>Type</b>	Qualitative
<b>Source</b>	Researcher
<b>Applicable Phase</b>	Requirement Analysis
<b>Purpose</b>	This measure is used in the experiment to determine the percentage of traceable requirement.
<b>Measurement Scale</b>	Interval
<b>Measurement Method</b>	Requirements in the requirement specification are individually read and checked for traceability. A requirement is traceable if it can be linked to its source and the related design, test and implementation [Davis 1993] [Ramesh and Jarke 2001]. Total number of traceable requirements are counted and recorded
<b>Related metrics</b>	Percentage of Traceable Requirements (Metric 1)

Table 6-15 Number of traceable requirements measure (Measure 1)

<b>Number of Requirements</b>	
<b>Definition</b>	Measures number of defined requirements
<b>Type</b>	Quantitative
<b>Source</b>	Experiment Subjects
<b>Applicable Phase</b>	Requirement Analysis
<b>Purpose</b>	This measure is used in the experiment to determine the value of the two metrics: Percentage of Traceable Requirements, and Test Case Density.
<b>Measurement Scale</b>	Interval
<b>Related metrics</b>	Percentage of Traceable Requirements (Metric 1), Test Case per Requirement Ratio (Metric 11)

Table 6-16 Number of requirements measure (measure 2)

### 6.3 Measurement Process Conduct

The designed measurement process required mechanisms for collecting and storing measurements. In this section, the data collection procedure conducted to capture and store measurement data, as well as tools used are discussed.

#### 6.3.1 Data Collection Procedure

In defining the data collection procedure, a number of issues were considered which are discussed in this section. Two methods of collecting and recording measurement were used:

1. **Measurements taken by the experiment subjects:** These measurements were taken by the subjects during the life of their projects and were submitted by them through a specific online measurement form (see Section 5.7.1). The measurement form contained a number of measurement questions, in accordance with the measurement requirements of the devised measurement process.
2. **Measurements taken by the researcher:** This was done after the completion of the projects and their assessments by the course tutors. The researcher studied and evaluated each project report with respect to a number of attributes (e.g. number of traceable requirements). This was done for both the treated and control projects.

During the course of the development projects, the subjects took measurements of various aspects of the project. The responsibility for collecting, recording, and submitting measurement data was assigned to the individual subjects in the case of individual projects, and to the groups in the case of group projects. The subject for the group projects (in both the treated and control groups) were instructed to take and record the data in their group meetings. Each group had a group leader, who was asked to ensure that measurement tasks were discussed at their group meetings and appropriate sections of the online measurement form were filled. The collected data were stored in a relational database for structured storage and provision of facilities to run queries on the data for analysis. This is further discussed in the Section 6.3.2.

The subjects were instructed to use an online data collection system, which was specifically designed and hosted on an internet website (see Appendix A. Experiment Details). The website hosted measurement forms that were to be filled in by the subjects. There were also guidelines and help on how to fill in the form. As a contingency plan, the subjects were also given the option of recording their measurement data manually in paper forms, if for some reason they could not use the online system. They were asked to include the completed form in their project report. The vast majority of the subjects used the online system to submit their measurements as instructed.

There were two issues related to the integrity and correctness of data. First, that the measurement data taken and recorded were correct, and second, that the data was transferred to the database correctly. In order to ensure that the data was correct and valid, the subjects were instructed to take and record the measurement data as carefully and precisely as they could. In the case of group projects, teams were encouraged to discuss the validity and



correctness of the measurements in their group meetings in order to reduce flaws. They were further instructed to be meticulous in completing the online measurement form. They were asked to recheck the values submitted and correct any possible mistakes. The small number of measurement data that were submitted on paper by the subjects with their project report, was inserted into the database twice by the researcher to eliminate the chance of incorrect data entering the database.

### 6.3.2 Tools Used

In a measurement programme, a number of tools can be used to facilitate the process. The number and type of the required tools, depends on the scope and sophistication of the measurement programme. It is essential that a suitable medium to facilitate the data storage and analysis of the collected measurements be provided. A web application was specifically devised to host the process patterns and the measurement forms (This was discussed in Section 5.7.1). The website included a relational database management system that was used for the storage of the submitted measurement data. The designed website and the included database for this measurement programme had the following characteristics:

- The website was designed to be simple and easy to use
- The database was designed to be flexible in case of any structural change requirements at later stages
- The database was normalised to avoid repetition of data
- The website and the included database incorporated security measures to prevent unauthorised access

The relational database was further important in systematically exporting the measurement data to a statistical package for analysis (i.e. SPSS package).

The employment of a suitable statistical analysis package to statistically analyse and present the results was important. Such tools further helped to analyse the results for correlations and statistical significance. In this research, the SPSS statistical analysis tool was used. This is further discussed in Chapter 7.

## 6.4 Summary

A goal-oriented strategy based on the GQM model was implemented in this study. The GQM model defines a practical way of implementing goal-oriented measurement. It introduced a mechanism for formulating goals for the measurement programme, as well as defining metrics required to achieve the goals. The measurement process involved the following main tasks:

1. Define goals
2. Define questions to answer to achieve the goals
3. Define metrics that provide answer to the questions
4. Define data collection procedure

For each development phase (i.e. Requirement Analysis, Design, Implementation, and Delivery), the three components of the GQM were defined through GQM tables, which specified the goals, questions, and the related metrics. A detailed specification and rationale for each involved metric was presented in individual tables, the complete set of which is presented in the 'Appendix C. Metrics Specifications'. For each metric, the data collection procedure and storage was also defined and described, where the measurements taken by the subjects were input to the system through online measurement forms hosted on a specific website. The measurement data was stored in a relational database by the devised web application, which was subsequently imported to the SPSS statistical package for analysis.

Having discussed the research methods (experimentation and measurement process) in the last two chapters, in the following chapter (Chapter 7) the experiment results will be presented and discussed.

---

## Chapter 7 Results

---

### 7.1 Introduction

In this chapter, the results of the experimental research method are presented and statistically analysed using the SPSS statistical analysis package. As discussed in Chapter 5, there were two types of projects involved in this experiment; *individual projects* and *group projects*. The experiment had two phases in the case of group projects (Semester 1, and Semester 2) and a single phase in the case of individual project (Semester 1). The experiment duration for each experiment phase was one semester. For both individual and group projects there were two distinct sources of measurement data, which were the measurement data collected through a measurement process, and the marks given to the completed projects by tutors. In this chapter, both sources of data are presented and analysed with respect to the experiment's objectives.

In the first section of the chapter, the results of the conducted measurements are presented and analysed, followed by a presentation and analysis of the results of the official marks in the latter section. The results of the treated groups' views, on the usefulness and usability of process patterns, will be presented in the final section. First, a brief discussion of the statistical methods employed is given in the following section.

### 7.2 Applied Statistical Methods

The first task in statistical analysis was to determine an appropriate statistical method for the analysis of the collected data. This involved determining whether the data was of parametric or non-parametric nature. This is discussed next.

#### 7.2.1 Parametric Vs Non-parametric

The parametric or non-parametric nature of the data determined whether parametric or non-parametric tests were appropriate for their analysis. An essential condition for parametric tests is that the data is *normally distributed*. Non-parametric tests are appropriate when one or more variables in the data set violates *the normality* assumption or the sample size is small (< 15 cases or subjects) [Moore and McCabe 1993]. Therefore, to determine which method was appropriate for the analysis of the experiment data, the data needed to be tested for *normal distribution*. This is generally done through investigating the statistics (Skewness and Kurtosis) of the data and their graphical representation such as histograms. More specifically, the Kolmogorov test [Ree2001] can be used to check for normality in data. It compares the scores in the sample to a set of normally distributed scores with the same mean and standard deviation. If the test was non-significant (i.e.  $p \geq 0.05$ ) then the distribution would be normal, otherwise (i.e.  $p < 0.05$ ) it would be non-normal. In order to test the data in the experiment for normality both methods (observation of graphical representations such as histograms, and Kolmogorov test) were employed. These tests indicated the presence of some outliers, which had to be dealt with in order for the data to pass the normal distribution test.

#### 7.2.2 Identification and Treatment of Outliers

An outlier is a score very different from the rest of the data that causes a distortion of statistics (e.g. sample means, variances) and, therefore, the results that include outliers often cannot be easily generalised. There are several other problematic effects of outliers [Field 2000] including inflated sums of squares, distortion of p-values bias or distortion of estimates, and faulty conclusions. There are mainly two likely causes of outliers, both of which were investigated in this research with respect to the experiment data. These are:

1. **Data errors:** These are caused by data recording or entry errors. All the outliers in the experiment were of this type.
2. **Rare event:** Data, that for some acceptable, reason does not fit within the typical range of other data values (e.g. A project with uncharacteristically large LOC)

There are a number of ways of detecting outliers [Johnson and Bhattacharyya 2001], two of which (inter-quartile-range (IQR) computation and z-score) were employed in this research. Boxplots, which were employed

in this research to identify outliers, use the inter-quartile-range (IQR) technique. Using a boxplot is an effective approach, especially when working with large datasets that have continuous data, such as the one in this research. Furthermore, boxplots make no distributional assumptions and depict extreme values, in a way that are easily identified. The other method employed for identifying outliers was the z-score technique. Since virtually all mound-shaped data falls within 3 standard deviations of its mean, the z-scores of such data are virtually all between -3 and 3. Thus any data that has a z-score less than (-3) or greater than (+3) would be an outlier. The z-score associated with the  $i^{\text{th}}$  observation of a random variable  $x$  is calculated as follows:

$$z_i = \frac{(x_i - \bar{x})}{s}$$

where

$$s = \sqrt{\frac{\sum_{i=1}^n (x_i - \bar{x})^2}{n - 1}}$$

Equation 7-1 z-score

The Application of the outlier detection techniques showed that there were a small number of outliers in the data set. There are a number of approaches proposed in the literature for resolving outlier problems, depending on the nature of the detected outliers [Barnett 1994]. The following resolution methods were considered for employment:

- **Correction:** Re-checking the source of the data for possible data entry errors and correcting them.
- **Transformation:** Transforming data (e.g. square roots and logarithms) is a way to soften the impact of outliers, since they shrink larger values to a much greater extent than they shrink smaller values. However, transformations may not fit into the theory of the model, or they may affect its interpretation (e.g. change the relationship between the original variable).
- **Deletion:** Only as a last resort should an outliers be deleted, and then only if it is found that they are legitimate errors that cannot be fixed, or lie so far outside the range of the remainder of the data that they distort statistical inferences.
- **Changing the score (Winsorising):** Use methods such as changing the score to be one unit above the next highest score in the data set.
- **Accommodation:** Using methods (i.e. non-parametric) that are robust in the presence of outliers.

Closer examination of these outliers indicated that the occurrences of the outliers were all due to incorrect data entry by the experiment subjects. In resolving the outliers, the corresponding subjects were consulted and the erroneous data was replaced by the correct values, thus resolving the outliers. An example of this was an entry of 105 for 'number of lines of code' variable, which proved to be an outlier. On consultation with the relevant subject, the true value was confirmed as 1050.

The parametric or non-parametric nature of the measurement data for the experiment variables were investigated, using the discussed process. The investigation showed that the measurement data for the experiment variables were *normally distributed*. Furthermore, the presence of a relatively large sample size (132 group projects, 128 individual projects) made the application of parametric tests to be justified. It was therefore appropriate to perform parametric, rather than non-parametric statistical tests.

### 7.2.3 Parametric Tests

Statistical analysis was employed in this experiment to evaluate and present the values of the variables and carry out statistical significance tests. Two methods of statistical analysis (i.e. Factorial Analysis of Variance (Two way ANOVA), and Independent Samples t-test) were used, depending on the circumstances and hypothesis being tested. These are discussed below.

#### 7.2.3.1 Independent Samples t-Test

The independent samples t-test is used when there is one dependent and one independent variable, where there are two experimental conditions (e.g. treated and control) and different subjects are assigned to each condition. It is employed to determine if there is a significant difference between the means of the dependent variable through

variations in the independent variable. As discussed above, the independent samples t-test is generally applicable to parametric test, where the data being tested is drawn from a normally distributed population with the same variance. However, the independent samples t-test is a robust method that stands up to some violation of these conditions [Howell 2002]. In this experiment, the independent samples t-test is employed to determine if there is a significant difference between the means of dependent variables (e.g. a metric such as Defect Density) for the independent variable *experiment-groups* (i.e. treated and control). This method is applied to the analysis of the individual projects, since only a single semester was involved. A significant difference in a dependent variable between the treated and control groups would indicate an effect of the treatment (i.e. process patterns). The independent samples t-test is calculated as follows [Rees 2001]:-

$$t = \frac{X_1 - X_2}{\sqrt{\left(\frac{SS_1 + SS_2}{n_1 + n_2 - 2}\right) \left(\frac{1}{n_1} + \frac{1}{n_2}\right)}}$$

Equation 7-2 Independent samples t-test

Where,  $X_1$  is the mean for group 1,  $X_2$  is the mean for group 2,  $SS_1$  is the sum of squares for group 1,  $SS_2$  is the sum of squares for group 2,  $n_1$  is the number of subjects in group 1, and  $n_2$  is the number of subjects in group 2.

There were, however, circumstances in the experiment where there were more than one independent variable concerned. There were two independent variables, *experiment group type* and semester, defined in the case of group projects. In such cases the independent samples t-test would not be appropriate and the 'Factorial ANOVA' (Two-way ANOVA) method is used for analysing the data. This is discussed next.

### 7.2.3.2 Two-way Analysis of Variance (ANOVA)

The two-way ANOVA (factorial ANOVA) examines the effects of multiple independent variables on one dependent variable. It provides the technique to assess the significance of the effect of two independent variables on a dependent variable, and determine any interaction between the independent variables. As discussed in the research method chapter (Chapter 5), the objective was to analyse the performance levels of the two experimental groups (*treated* and *control*) with respect to a number of software attributes. This was done across two semesters, where in each semester different subjects acted as treated or control groups. There were therefore two independent variables, which were the experiment group-type (i.e. treated and control groups), and the semester (i.e. semester 1 and semester 2). There was also a single dependent variable, which was the software attribute (e.g. metric or tutor mark) being evaluated. The relationships between the experiment groups and semesters are illustrated Table 7-1.

The aim was to determine whether there were any significant differences between the means of the groups across each independent variable. That is, whether there was a significant difference in the performance levels of: a) experimental and control groups, and b) *Semester 1* and *semester 2* subjects. The aim was to determine if there was any significant difference between mean values in the treated and control groups for any metric/mark, and whether the difference in semesters affected the outcome.

<b>Phase 1</b> = Semester One		
<b>Phase 2</b> = Semester Two		
√ = Treated		
× = Control		
<b>TP Campus</b> = Subjects taking module CMT3991 at Trent Park Campus		
<b>HE &amp; TM Campuses</b> = Subjects taking module CMT3991 at Hendon or Tottenham sites		
	<b>Phase 1</b>	<b>Phase 2</b>
<b>TP Campus</b>	√	×
<b>HE &amp; TM Campuses</b>	×	√

Table 7-1 Relationships between experiment groups and semesters

The significance value ( $p$ ) for the measurements of the independent variable, with respect to the dependent variable, were evaluated and depicted in relevant tables in each section. The value- $P$  represents the probability that the difference between the means of the groups examined could have happened by chance. Conventionally, it has to be less than 0.05 for statistical significance for a confidence level of 95%.

## 7.3 Teams Vs Individuals

In this experiment, two types of projects were under examination, group projects and individual projects. As well as investigating any difference between the treated and control groups within each project type, this study investigates whether the effect of process patterns was more prominent on group projects than on individual projects. In investigating any significant difference in performance between the group-projects (teams) and individual-projects (individuals), a two-way Analysis of Variance (two-way ANOVA) was used to determine the effect of *experiment project-type* on all the values of the investigated software attributes. In this case, there were two independent variables, the experiment group types (i.e. treated and control) and project types (i.e. individual projects and group projects) and a single dependent variable, which was the attribute being examined (e.g. defect density).

### 7.3.1 Further Analysis

The results were further statistically analysed for significance, irrespective of project types and experiment phases (i.e. semesters). In this analysis, both group and individual projects were considered collectively, without taking into account their project type or the semester in which they were conducted. Although this analysis is rather generic, without differentiating between group and individual projects, it has the advantage of a larger sample size of 260 (132 of which are group projects and 128 individual projects).

The statistical analysis method used for this analysis is the independent samples t-test, discussed above, where, in this case, the independent variable was the (experiment group type) and the dependent variable was the metric or tutor mark being investigated. The results in this analysis were in line with those achieved where group and individual projects were analysed individually and separately. Results of this analysis are presented in the Appendix D. Results.

## 7.4 Sensitivity Analysis

Sensitivity analysis provides the means of evaluating sensitivities of measures with respect to parameters of interest. It is, in general, a technique for determining the behaviour of a system by successively changing input values by a small amount and determining the changes in the outputs. However, for measurement data and analysis, sensitivity analysis is defined more broadly. Lang and Secic [1997] and Kitchenham et al. [2002b] propose the following tasks for sensitivity analysis before moving on to the statistical techniques:

1. Identify and treat outliers
2. Ensure that the data does not violate the assumptions of the tests used on them
3. Apply appropriate quality control procedures to verify your results

These guidelines were followed, as discussed earlier in this chapter, where the measurement data was analysed for outliers and distribution types (i.e. normal, non-normal) and their suitability for parametric statistical analysis.

Having carried out the initial steps of sensitivity analysis, the next step was to perform the mathematical and statistical procedures. This is generally performed by varying the system's parameters systematically, by a small fixed percentage, so that the relative impacts of each parameter could be directly compared [Saltelli et al. 2004].

Given the following linear equation:

$$Y = \sum_{i=1}^r \Omega_i Z_i$$

Where  $Y$  is the output,  $\Omega_i$  are fixed coefficients and  $Z_i$  are the uncertain input factors with the following distribution:

$$Z_i \approx N(\bar{Z}_i, \sigma_{Z_i}), \bar{Z}_i = 0, i = 1, 2, \dots, r.$$

Sensitivity analysis is then calculated to be the derivative, such as:  $S_{Z_i}^d = \frac{\partial Y}{\partial Z_i} = \Omega_i$ .

The goal in sensitivity analysis is to show the effects of changing parameter values. The sensitivity,  $s = \frac{\Delta o}{\Delta i}$ , is the amount of change in its output  $\Delta o$  that occurs due to a change to input by amount  $\Delta i$ . This was the method adopted to perform sensitivity analysis on the metrics and tutor marks in this research. By iteratively and continuously varying metric/mark values by a small percentage, the change in the outcome was measured and checked for statistical significance in each iteration. The objective of the sensitivity analysis here was to determine the amount, by which the parameters can change, before voiding and nullifying the statistical significance of the results (i.e. metric results would no longer be statistically significant) [Wakefield 2004].

In carrying out sensitivity analysis on metrics in this research, the value of each metric and tutor mark was changed by a small percentage (0.1%) at a time, and the result was evaluated for statistical significance. This simulation was carried out iteratively and systematically using the SPSS statistical package. The percentage of change that could be tolerated (before affecting the statistical significance of the results) was the sensitivity margin of the metric or the tutor mark. The sensitivity analysis result, for each metric and mark, is presented in the results in this chapter. This value is presented as the 'sensitivity margin', which is the percentage by which the metric/mark value could change before the results would no longer be statistically significant.

### 7.5 Correlation/Regression Analysis

Correlation and regression analysis deal with relationships among variables and were employed here to determine the correlation between pattern usage and metric/tutor mark values. The correlation coefficient  $r$  (referred to as Pearson linear correlation) is a measure of the linear relationship between two variables or attributes, and is defined as the ratio of the covariance of the sample populations to the product of their standard deviations. Values of  $r$  are always between -1 and +1, where ( $r=+1$ ) and ( $r=-1$ ) indicates that two variables are perfectly related in a positive or negative linear relationship respectively. A correlation coefficient of zero (i.e.  $r = 0$ ) indicates that there is no correlation between the two variables. Given two  $n$ -element sample populations,  $X$  and  $Y$ ,  $r$  is calculated as in Equation 7-3.

$$r = \frac{\text{covariance of X and Y}}{(\text{standard deviation of X})(\text{standard deviation of Y})}$$

$$r = \frac{\frac{1}{N-1} \sum_{i=0}^{N-1} \left( x_i - \left[ \frac{\sum_{k=0}^{N-1} x_k}{N} \right] \right) \left( y_i - \left[ \frac{\sum_{k=0}^{N-1} y_k}{N} \right] \right)}{\sqrt{\frac{1}{N-1} \sum_{i=0}^{N-1} \left( x_i - \left[ \frac{\sum_{k=0}^{N-1} x_k}{N} \right] \right)^2} \sqrt{\frac{1}{N-1} \sum_{i=0}^{N-1} \left( y_i - \left[ \frac{\sum_{k=0}^{N-1} y_k}{N} \right] \right)^2}}$$

Equation 7-3 Correlation coefficient

The correlation is high, if it can be 'summarised' by a straight line (sloped upwards or downwards), called the *regression line* of the form  $f(x) = \beta_0 + \beta_1 x$  (depicted in Figure 7-1 and Figure 7-2), and the response variable  $Y$  is modelled as:  $Y = f(X) + \epsilon = \beta_0 + \beta_1 x + \epsilon$ , where the random noise  $\epsilon$  is assumed to have normal distribution  $N(\sigma, \sigma^2)$ .

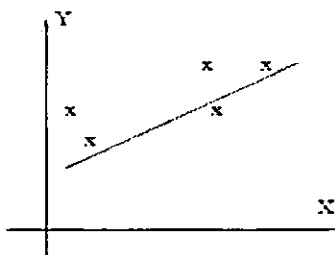


Figure 7-1 Regression line

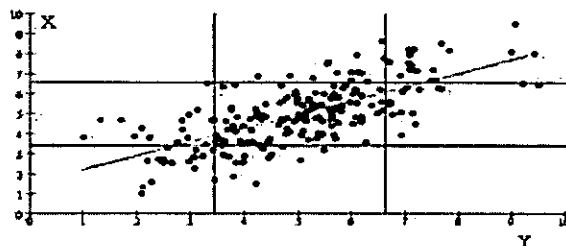


Figure 7-2 Regression line scatter plot

The correlation between the rate of treatment usage and the value of metrics/marks would indicate whether the higher rate of pattern usage corresponded to higher values of the metrics/marks. The pattern rate of usage was measured by the number of times the subjects logged in to the online treatment resource (see Section 5.7.1),

where the process patterns were hosted. The SPSS statistical analysis package tool was used to calculate the correlation coefficient, the correlation statistical significance and to produce scatter plots of the correlation for all metrics and marks for the group and individual projects. The results are presented in the results section of each metric/mark in this chapter.

### 7.5.1 Treatment Rate of Usage

As discussed in the research method chapter (Chapter 5), the process patterns (i.e. treatment condition) were hosted online through a website, to which only the treated groups had. The Figure 7-3 presents the average number of logins for both types of projects during the 12 weeks semesters. In the case of group projects there were two 12 weeks semesters whose logins are averaged.

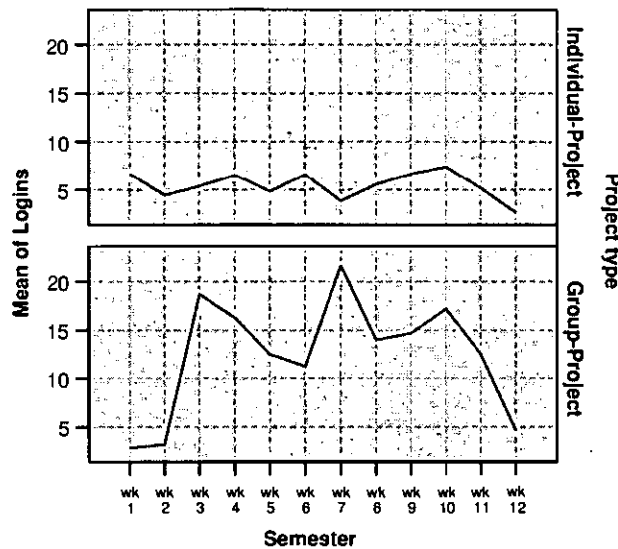


Figure 7-3 Rate logins to the treatment (i.e. process patterns) website

There were an average total of 148 logins for each group projects, and 65 logins for each individual projects. The results indicate that, whilst there had been continuous and regular usage of process patterns for both group and individual projects, there had been a substantially higher number of hits registered for group projects than the individual projects. This is partially due to the number of members in the group projects. The other reason is that, the students who took the individual projects had already completed the group projects, and had therefore accessed the online patterns in the previous semester.

## 7.6 Conducted Measurement Results

The dependent variables to be tested are a number of software attributes in the following four major development lifecycle phases through a number of metrics as discussed in Chapter 6:

- Requirement Analysis (RA)
- Design
- Implementation
- Delivery

In this section the results of the metrics, for each development phase, are statistically analysed and presented using the SPSS statistical package. The results of the metrics are presented in a number of tables and figures. For each metric, there is a table that states the means and standard deviations, and a boxplot that depicts the range and median values of the metrics with respect to project types, semesters, and experimental group types. A further table for each metric presents the statistical significance analysis of the metric.

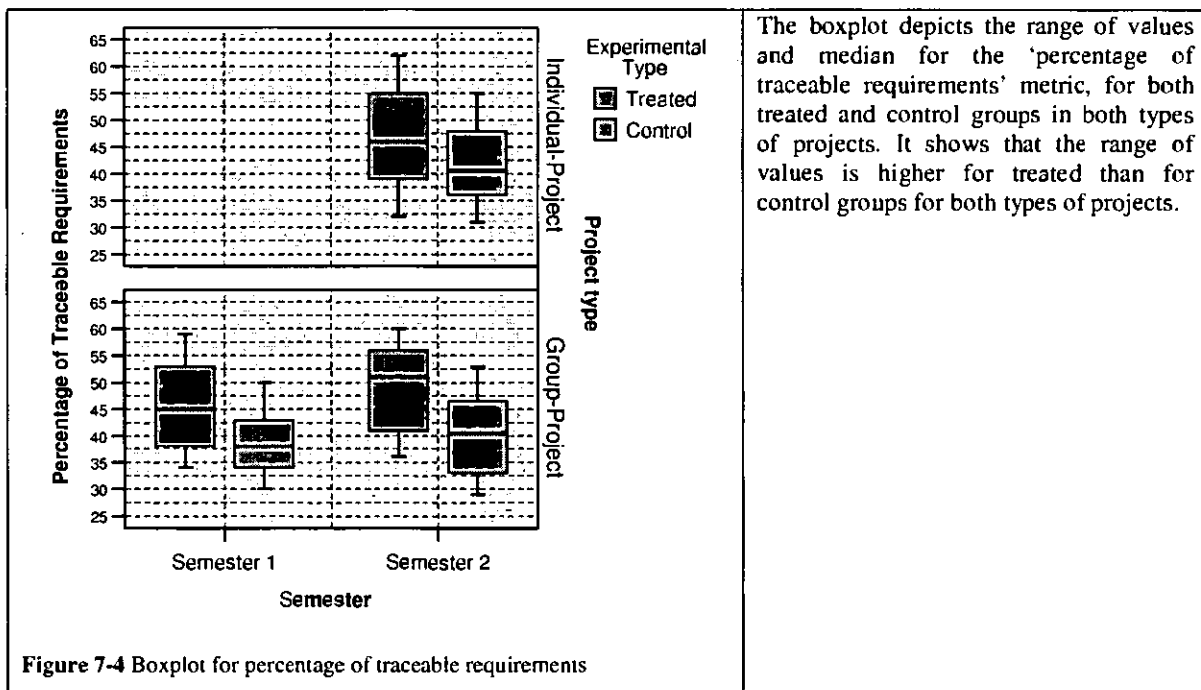
### 7.6.1 Requirements Analysis Phase

In this section the result of the metrics for the first development phase (i.e. Requirement Analysis) are presented. For the details and specification of these and all other metrics in this research, see 'Appendix C. Metrics Specifications'. The results of the following metrics are presented in this section.

- Percentage of traceable requirements
- Percentage of reviewed requirements specification
- Percentage of defects fixed
- Percentage of (RA) phase time spent in testing

Project type	Experimental Type	Semester	Mean	N (no. of cases)	Std. Deviation
Group-Project	Treated	Semester 1	45.85	47	7.743
		Semester 2	49.72	18	7.560
	Control	Semester 1	38.74	47	5.739
		Semester 2	40.35	20	7.775
Individual-Project	Treated	Semester 2	46.97	66	9.217
	Control	Semester 2	41.84	62	7.069

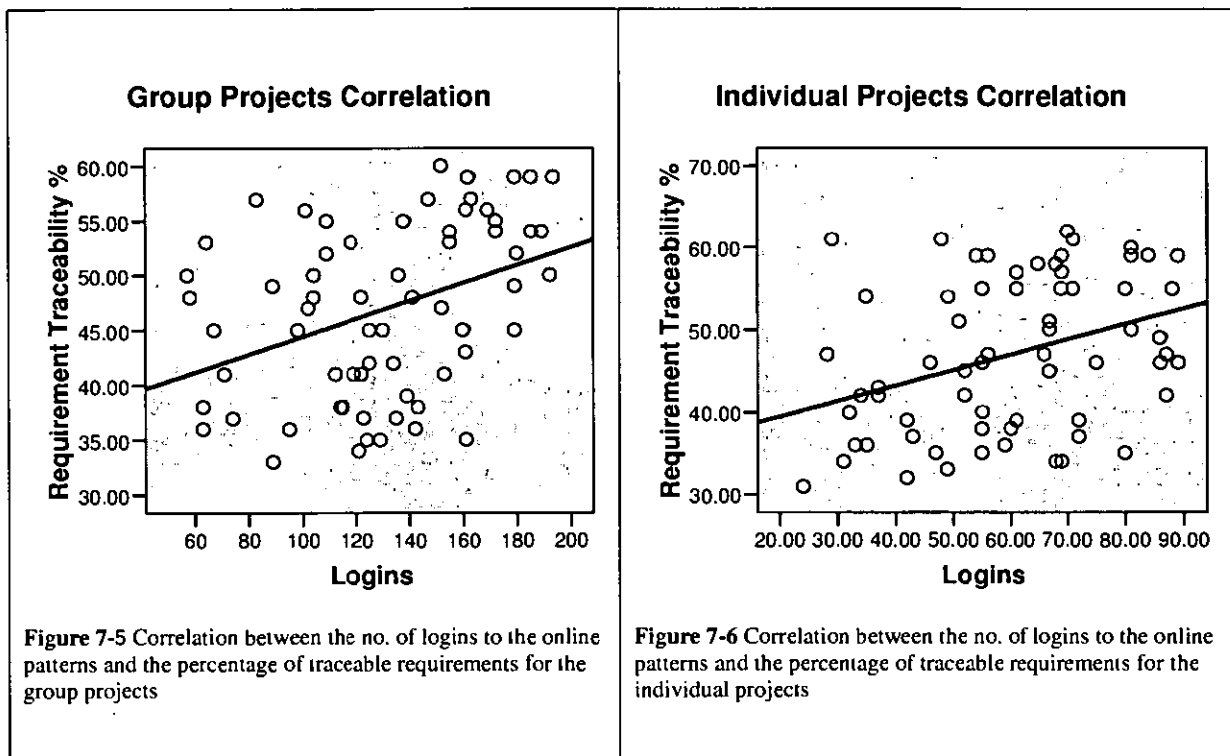
Table 7-2 Statistics for percentage of traceable requirements



The boxplot depicts the range of values and median for the 'percentage of traceable requirements' metric, for both treated and control groups in both types of projects. It shows that the range of values is higher for treated than for control groups for both types of projects.

Figure 7-4 Boxplot for percentage of traceable requirements





#### Percentage of Traceable requirement

The results of the statistical analysis to determine the statistical significance of any difference between treated and control groups for each project type, are as follows:

##### Group Projects

A  $2 \times 2$  independent measure ANOVA was carried out using the *experiment group-types* (i.e. treated and control) and the *semester* (i.e. semester-one and semester-two), as the independent variables, and the 'percentage of traceable requirements' as the dependent variable. There was a main effect for *experiment group-type*, with treated groups scoring significantly better than control groups (i.e.  $p < 0.05$ ),  $p = 0.006$ . The main effect for the *semester* was not significant (i.e.  $p \geq 0.05$ ),  $p = 0.813$ . The main effect for the interaction between *experiment group-type* and *semester* was also not significant (i.e.  $p \geq 0.05$ ),  $p = 0.45$ .

The sensitivity analysis showed the sensitivity margin to be 19.1%.

There was a statistically significant positive correlation between this metric and 'logins'  $r = 0.32$ ,  $p = 0.030$  as depicted in the scatter plot Figure 7-5.

##### Individual Projects

The mean difference in percentage of traceable requirements for treated and control groups were compared using an independent samples t-test. The result showed that there was a mean difference between the treated and control groups, for the percentage of traceable requirements, which was statistically significant (i.e.  $p < 0.05$ ) at  $p = 0.011$ .

The sensitivity analysis showed the sensitivity margin to be 11.1%.

There was a statistically significant correlation between this metric and 'logins'  $r = 0.44$ ,  $p = 0.009$  as depicted in the scatter plot Figure 7-6.

##### Concluding Remarks

There was a mean difference between the treated and control groups, in both individual and group projects for this metric, which was shown to be statistically significant. This indicates that treatment (i.e. use of process patterns) was effective.

The results of the statistical analysis to determine the statistical significance of the effect of project type on the experiment group-types are as follows:

**Group Projects & Individual projects**

A 2 × 2 independent measure ANOVA was carried out using the two variables, *experiment group-types* (i.e. treated and control) and the *project-type* (i.e. Individual project, and Group projects) as the independent variables, and the *percentage of traceable requirements* as the dependent variable. There was a main effect for *experiment group-type*, with treated groups scoring significantly better than control groups (i.e.  $p < 0.05$ ),  $p = 0.002$ . The main effect for the *project-type* was significant (i.e.  $p < 0.05$ ),  $p = 0.023$ . The main effect for the interaction between *experiment group-type* and *project type* was also significant (i.e.  $p < 0.05$ ),  $p = 0.014$ .

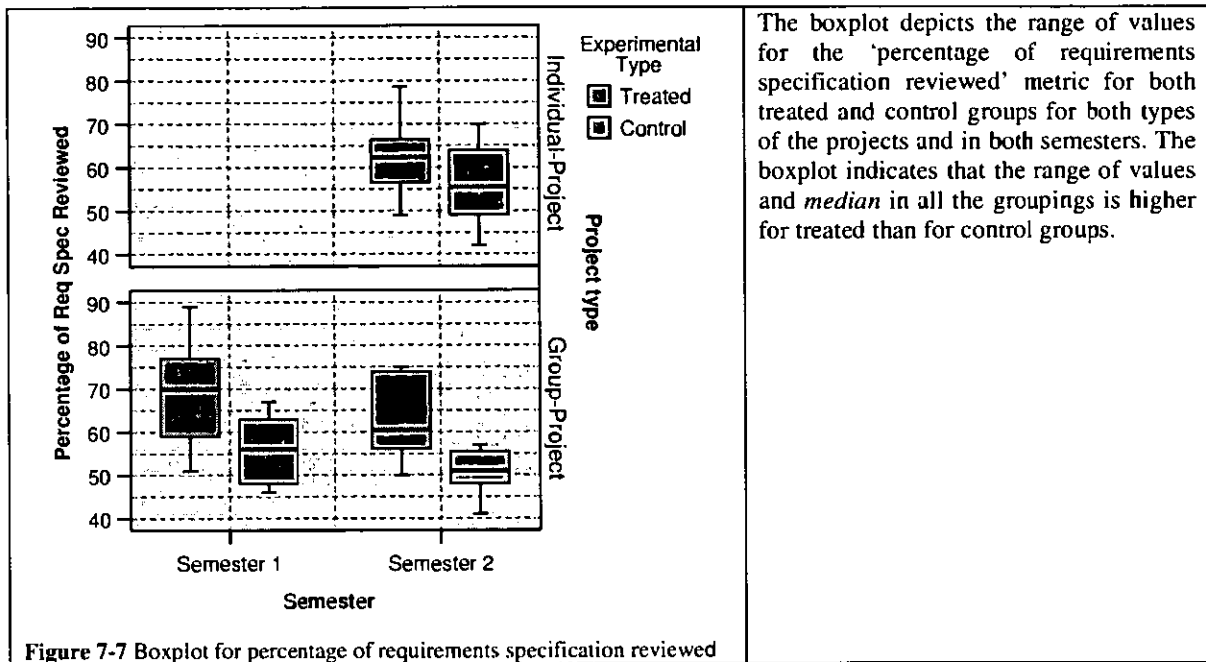
**Concluding Remarks**

There is a statistically significant difference between group and individual projects, in terms of the mean difference between the treated and control groups, for this metric. This indicates that the treatment (i.e. use of process patterns) was more effective on group projects than on individual projects for this metric.

<b>Metric Result Summary</b>	
<b>Percentage of traceable requirements</b>	
<p>The results (depicted in Table 1-1 and Figure 7-4) show that treated groups had a higher percentage of traceable requirements than the control groups for both group and individual projects which was shown to be statistically significant. The results show that requirement traceability was improved by a sensitivity margin of 19.1% in group projects and 11.1% in individual projects. The results, therefore, indicate that the use of process patterns has a significant positive effect in increasing the percentage of traceable requirements. Based on these results, it can therefore be deduced that the application of process patterns improves the requirements traceability.</p>	
<p>It has been further shown that the mean difference between the treated groups and control in the group projects, in terms of the percentage of traceable requirements, was statistically significantly higher than in the individual projects. This indicates that the employment of process patterns has been more effective in group projects than in individual projects for this metric. It can therefore be deduced that process patterns are more effective on group projects than individual projects in improving traceability requirements.</p>	

Project type	Experimental Type	Semester	Mean	N (no. of cases)	Std. Deviation
Group-Project	Treated	Semester 1	69.34	47	11.495
		Semester 2	62.56	18	8.959
	Control	Semester 1	56.28	47	7.228
		Semester 2	51.00	20	4.768
Individual-Project	Treated	Semester 2	62.59	66	7.324
	Control	Semester 2	55.90	62	8.358

Table 7-3 Statistics for the percentage of the requirements specification reviewed



The boxplot depicts the range of values for the 'percentage of requirements specification reviewed' metric for both treated and control groups for both types of the projects and in both semesters. The boxplot indicates that the range of values and *median* in all the groupings is higher for treated than for control groups.

Figure 7-7 Boxplot for percentage of requirements specification reviewed

A full and detailed statistical analysis textual report (as presented for the previous metric) on each metric would make this report too excessive in size. Therefore, for the remaining analysed metrics/marks, a brief summary of the results of the performed statistical analysis is presented in table format. The table's main layout and sections is depicted in Table 7-4 below.

No.	Elements
1	Metric title
2	Statistical analysis of group project
3	Statistical analysis of individual project
4	Statistical analysis of difference between group and individual projects
5	Correlation analysis between number of logins and the metric value

Table 7-4 Results tables layout

Percentage of Reviewed Requirements Specification			
Statistical significance analysis of mean difference between treated and control groups			
Operation	Independent Variables	Dependent Variable	Projects
2 × 2 independent measure: ANOVA	Experiment group-type Semester	Percentage of Reviewed Requirements Specification	Group Projects
Significance p-value	Description	Sensitivity Margin (%)	
0.002	There is a main effect for <i>experiment group-type</i>	20.6	
0.883	The main effect for the <i>semester</i> is not significant		
0.543	The main effect for the interaction is not significant		
Operation	Independent Variable	Dependent Variable	Projects
Independent Samples t-test	Experiment group-type	Percentage of Reviewed Requirements Specification	Individual Projects
Significance p-value	Description	Sensitivity Margin (%)	
0.013	There is a main effect for <i>experiment group-type</i>	10.1	
<b>Conclusion</b>	There is a statistically significant difference between the treated and control groups in both individual and group projects for this metric. This indicates that the treatment (i.e. use of process patterns) was statistically effective.		
Statistical significance analysis of the effect of project type on the experiment group-types			
Operation	Independent Variables	Dependent Variable	Projects
2 × 2 independent measure ANOVA	Experiment group-type Project-type	Percentage of Reviewed Requirements Specification	Group Projects Individual projects
Significance p-value	Description		
0.006	There is a main effect for <i>experiment group-type</i>		
0.010	The main effect for the <i>group-type</i> is significant		
0.011	The main effect for the interaction is significant		
<b>Conclusion</b>	There is a statistically significant difference between group and individual projects in terms of the mean difference between the treated and control groups for this metric. This indicates that the treatment (i.e. use of process patterns) was statistically more effective on group projects than on individual projects for this metric.		
Correlation Analysis for the 'no. of logins' and 'this metric' for group and individual projects			
Correlation Coef. (r)	Significance (P)	Description	Project type
0.372	0.011	There is a statistically significant positive correlation	Group Projects
0.317	0.020	There is a statistically significant positive correlation	Individual Projects

Table 7-5 Statistical analysis for the 'percentage of reviewed requirements specification' metric

<b>Metric Result Summary</b>
Percentage of reviewed requirements specification
The results (depicted in Table 7-3, Table 7-5, and Figure 7-7) show that treated groups reviewed a higher percentage of the requirements specification than the control groups (for both group and individual projects) which was shown to be statistically significant. The results show that the requirements specification review was improved by a sensitivity margin of 20.6% in group projects and 10.1% in individual projects. The results indicate that the use of process patterns had a positive effect on the percentage of the requirements specification reviewed. Based on these results, it can therefore be deduced that the application of process patterns improves the requirements specification reviews.
It has been further shown that the mean difference between the treated groups and control groups in group projects, in terms of the percentage of reviewed requirements specification, was significantly higher than in the individual projects. This indicates that the employment of process patterns has been more effective in group projects than in individual projects for this metric. It can therefore be deduced that process patterns are more effective on group projects than individual projects in improving requirements specification reviews.

Project type	Experimental Type	Semester	Mean	N (no. of cases)	Std. Deviation
Group-Project	Treated	Semester 1	83.68	47	10.867
		Semester 2	84.06	18	9.545
	Control	Semester 1	83.60	47	10.137
		Semester 2	80.25	20	10.249
Individual-Project	Treated	Semester 2	80.85	66	10.458
	Control	Semester 2	79.77	62	9.723

Table 7-6 Statistics for the percentage of defects fixed in RA phase

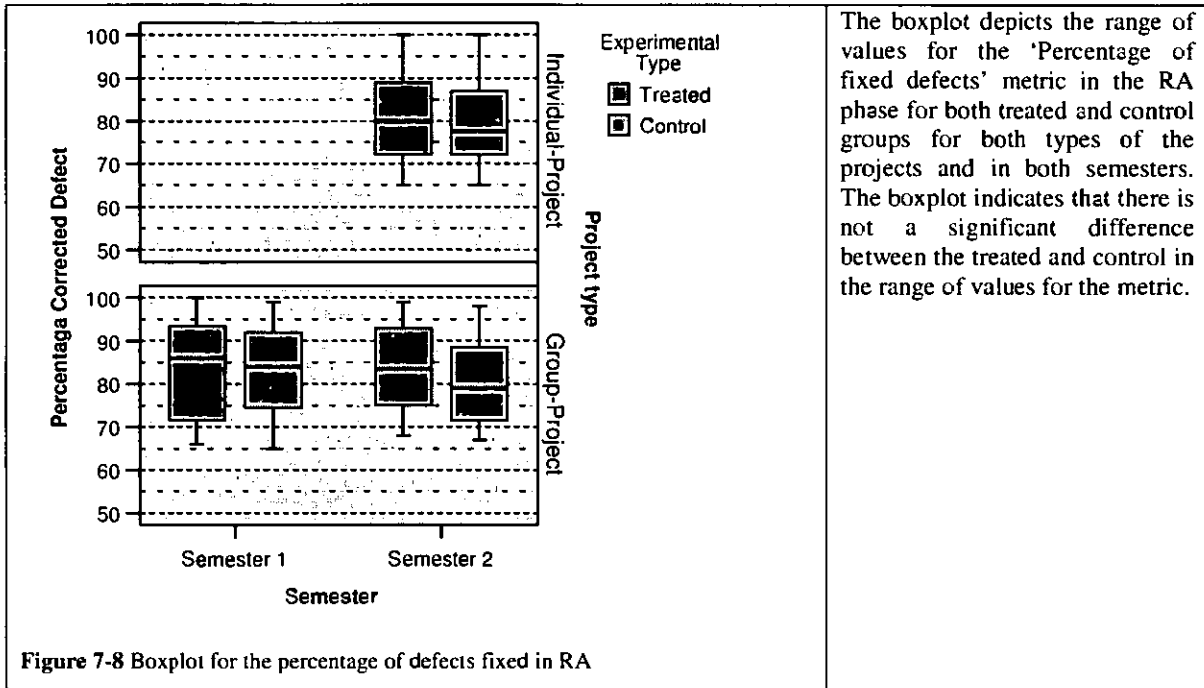


Figure 7-8 Boxplot for the percentage of defects fixed in RA

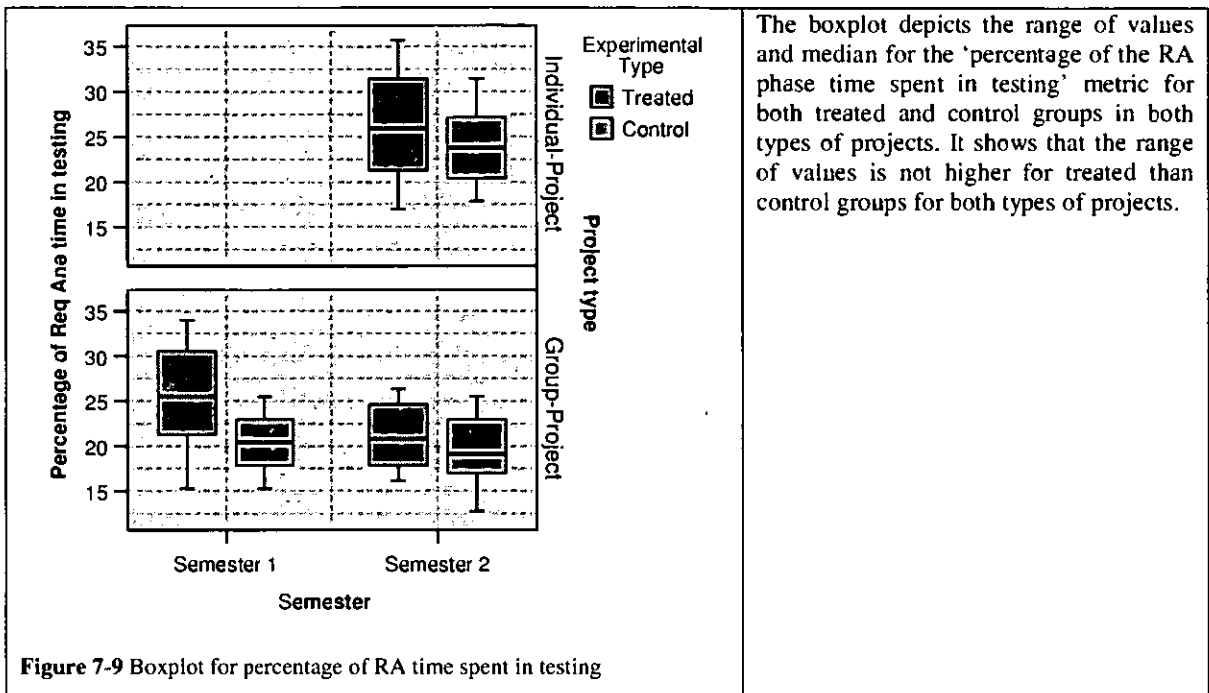
Percentage of Defects Fixed (RA phase)			
Statistical significance analysis of mean difference between treated and control groups			
Operation	Independent Variables	Dependent Variable	Projects
2 × 2 independent measure ANOVA	Experiment group-type Semester	Percentage of defects fixed (RA phase)	Group Projects
Significance p-value	Description		Sensitivity/Margin (%)
0.143	The main effect for the experiment group-type was not significant		N/A
0.783	The main effect for the semester was not significant		(No significant difference between treated and control groups)
0.511	The main effect for the interaction was not significant		
Operation	Independent Variable	Dependent Variable	Projects
Independent Samples t-test	Experiment group-type	Percentage of defects fixed (RA phase)	Individual Projects
Significance p-value	Comment		Sensitivity/Margin (%)
0.144	There main effect for experiment group-type was not significant		N/A (No significant difference between treated and control groups)
<b>Conclusion</b>	There is not a statistically significant difference between the treated and control groups in either individual or group projects for this metric. This indicates that treatment (i.e. use of process patterns) was not statistically effective.		
Statistical significance analysis of the effect of project type on the experiment group-types			
Operation	Independent Variables	Dependent Variable	Projects
2 × 2 independent measure ANOVA	Experiment group-type Project-type	Percentage of defects fixed	Group Projects Individual projects
Significance p-value	Comment		
0.234	The main effect for the experiment group-type was not significant		
0.213	The main effect for the group-type was not significant		
0.463	The main effect for the interaction was not significant		
<b>Conclusion</b>	There is not a statistically significant difference between group and individual projects in terms of the mean difference between the treated and control groups for this metric. This indicates that treatment (i.e. use of process patterns) was not statistically more effective on either group or individual projects for this metric.		
Correlation Analysis for the 'no. of logins' and 'this metric' for group and individual projects			
Correlation Coef. (r)	Significance (P)	Description	Project type
0.081	0.270	There is not a statistically significant positive correlation	Group Projects
0.101	0.291	There is not a statistically significant positive correlation	Individual Projects

Table 7-7 Statistical significance analysis for the 'percentage of defects fixed' metric

<i>Metric/Result/Summary</i>	
Percentage of Defects Fixed (RA phase)	
<p>The results (depicted in Table 7-6, Table 7-7, and Figure 7-8 ) show that there is not a statistically significant mean difference between the treated and control groups, in terms of the percentage of defects fixed, for both group and individual projects. Based on these results, it can therefore be deduced that the use of process patterns does not significantly increase the percentage of the defects fixed in the Requirement Analysis phase.</p> <p>It has been further shown that the difference between the treated groups and control in group projects, in terms of the percentage of defects fixed, was not significantly different to those in the individual projects. It can therefore be deduced that process patterns are not more effective in group projects than in individual projects for this metric.</p>	

Project type	Experimental Type	Semester	Mean	N (no. of cases)	Std. Devlition
Group-Project	Treated	Semester 1	22.63	47	5.647
		Semester 2	21.49	18	3.496
	Control	Semester 1	20.47	47	2.928
		Semester 2	19.59	20	3.643
Individual-Project	Treated	Semester 2	26.20	66	5.699
	Control	Semester 2	24.12	62	3.964

Table 7-8 Statistics for the percentage of RA phase time spent in testing



The boxplot depicts the range of values and median for the 'percentage of the RA phase time spent in testing' metric for both treated and control groups in both types of projects. It shows that the range of values is not higher for treated than control groups for both types of projects.

Figure 7-9 Boxplot for percentage of RA time spent in testing

Percentage of Phase Time Spent in Testing (RA phase)			
Statistical significance analysis of mean difference between treated and control groups			
Operation	Independent Variables	Dependent Variable	Projects
2 × 2 independent measure ANOVA	Experiment group-type Semester	Percentage of RA Phase Time Spent in Testing	Group Projects
Significance p-value	Description	Sensitivity Margin (%)	
0.353	The main effect for the experiment group-type is not significant	N/A	
0.494	The main effect for the semester is not significant	(No significant difference between treated and control groups)	
0.264	The main effect for the interaction is not significant		
Operation	Independent Variable	Dependent Variable	Projects
Independent Samples t-test	Experiment group-type	Percentage of RA Phase Time Spent in Testing	Individual Projects
Significance p-value	Comment	Sensitivity Margin (%)	
0.103	There main effect for experiment group-type is not significant	N/A (No significant difference between treated and control groups)	
<b>Conclusion</b>	There is not a statistically significant difference between the treated and control groups in either individual or group projects for this metric. This indicates that treatment (i.e. use of process patterns) was not statistically effective.		
Statistical significance analysis of the effect of project type on the experiment group-types			
Operation	Independent Variables	Dependent Variable	Projects
2 × 2 independent measure ANOVA	Experiment group-type Project-type	Percentage of RA Phase Time Spent in Testing	Group Projects Individual projects
Significance p-value	Comment		
0.234	The main effect for the experiment group-type is not significant		
0.213	The main effect for the group-type is not significant		
0.463	The main effect for the interaction is not significant		
<b>Conclusion</b>	There is not a statistically significant difference between group and individual projects in terms of the mean difference between the treated and control groups for this metric. This indicates that treatment (i.e. use of process patterns) was not statistically more effective on either group or individual projects for this metric.		
Correlation Analysis for the 'no. of logins' and 'this metric' for group and individual projects			
Correlation Coef. (r)	Significance (P)	Description	Project type
0.113	0.123	There is not a statistically significant positive correlation	Group Projects
0.104	0.110	There is not a statistically significant positive correlation	Individual Projects

Table 7-9 Results of significance analysis for the 'percentage of RA phase time spent in testing' metric

<i>Metric Result Summary</i>
Percentage of Phase Time Spent in Testing (RA phase)
The results (depicted in Table 7-8, Table 7-9, and Figure 7-9) show that there was not a statistically significant mean difference between the treated and control groups in terms of the proportion of RA phase time spent in testing. Based on these results, it can be deduced that the application of process patterns did not significantly affect the proportion of RA phase time spent in testing.
It has been further shown that the mean difference between the treated groups and control groups in group projects, in terms of the percentage of phase time spent on testing, was not significantly different to those in the individual projects. It can therefore be deduced that process patterns are not more effective in group projects than in individual projects for this metric.

### 7.6.2 Design Phase

In this section the result of the metrics in the second development phase (i.e. design) are presented. The following metrics are analysed:

- Percentage of the design document reviewed
- No. of methods per class (Methods per Class ratio)



- Percentage of defects fixed
- Percentage of Design phase time spent in testing

Project type	Experimental Type	Semester	Mean	N(no. of cases)	Std. Deviation
Group-Project	Treated	Semester 1	42.51	47	7.793
		Semester 2	42.06	18	7.502
	Control	Semester 1	31.11	47	6.623
		Semester 2	28.35	20	7.361
Individual-Project	Treated	Semester 2	38.46	66	7.484
	Control	Semester 2	28.28	62	6.111

Table 7-10 Statistics for the percentage of design document reviewed

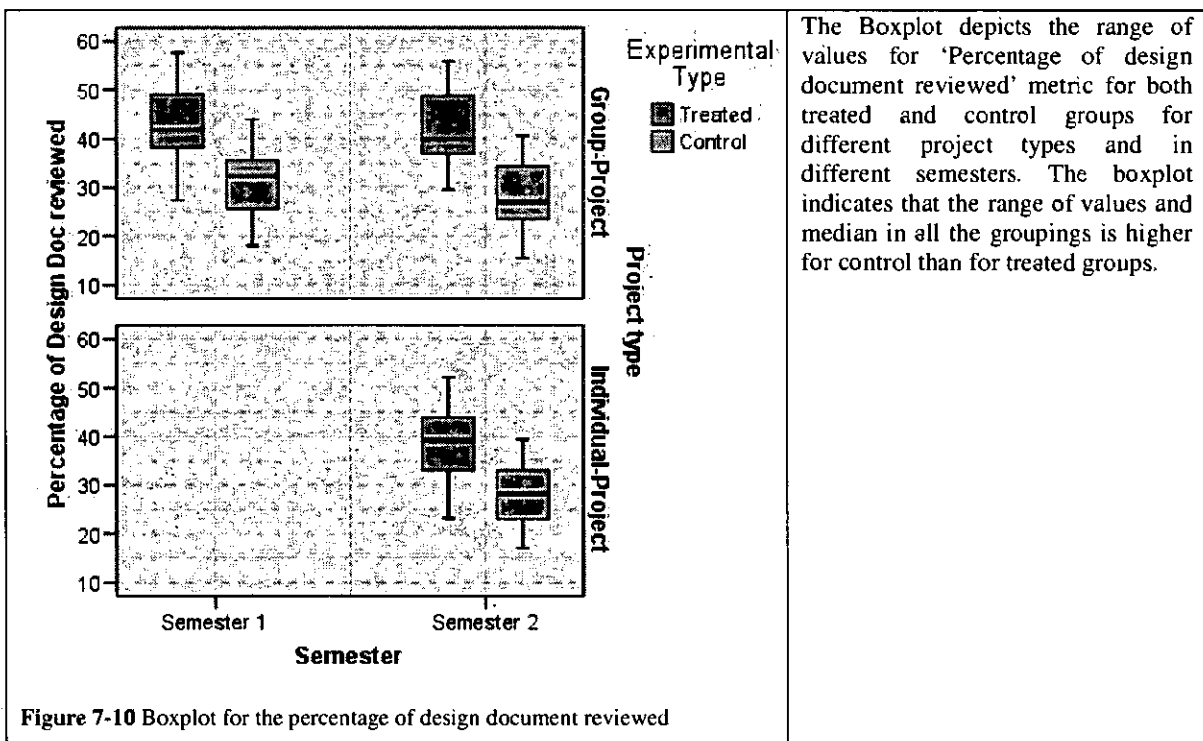


Figure 7-10 Boxplot for the percentage of design document reviewed

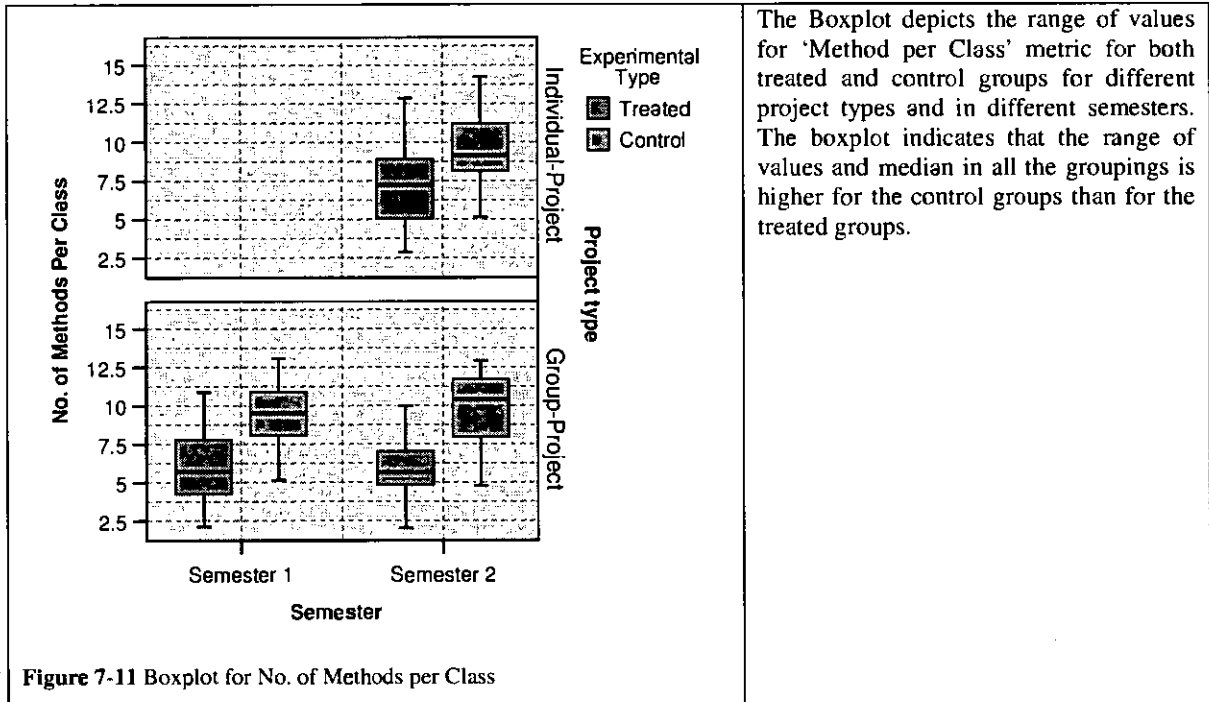
Percentage of Design Document Reviewed			
Statistical significance analysis of mean difference between treated and control groups			
Operation	Independent Variables	Dependent Variable	Projects
2 × 2 independent measure ANOVA	Experiment group-type Semester	Percentage of Design Document Reviewed	Group Projects
Significance p-value	Description		Sensitivity Margin (%)
0.001	There is a main effect for <i>experiment group-type</i>		38.1
0.686	The main effect for the <i>semester</i> is not significant		
0.419	The main effect for the interaction is not significant		
Operation	Independent Variable	Dependent Variable	Projects
Independent Samples t-test	Experiment group-type	Percentage of Design Document Reviewed	Individual Projects
Significance p-value	Description		Sensitivity Margin (%)
0.004	There is a main effect for <i>experiment group-type</i>		32.1
<b>Conclusion</b>	There is a statistically significant difference between the treated and control groups in both individual and group projects for this metric. This indicates that the treatment (i.e. use of process patterns) was statistically effective.		
Statistical significance analysis of the effect of project type on the experiment group-types			
Operation	Independent Variables	Dependent Variable	Projects
2 × 2 independent measure ANOVA	Experiment group-type Project-type	Percentage of Design Document Reviewed	Group Projects Individual projects
Significance p-value	Description		
0.001	There is a main effect for <i>experiment group-type</i>		
0.011	The main effect for the <i>group-type</i> is significant		
0.014	The main effect for the interaction is significant		
<b>Conclusion</b>	There is a statistically significant difference between group and individual projects in terms of the mean difference between the treated and control groups for this metric. This indicates that the treatment (i.e. use of process patterns) was statistically more effective on group projects than on individual projects for this metric.		
Correlation Analysis for the 'no. of logins' and 'this metric' for group and individual projects			
Correlation Coef. (r)	Significance (P)	Description	Project type
0.343	0.001	There is a statistically significant positive correlation	Group Projects
0.321	0.002	There is a statistically significant positive correlation	Individual Projects

Table 7-11 Statistical significance analysis for the percentage of design documents reviewed metric

<b>Metric Result Summary</b>
Percentage of Design Document Reviewed
<p>The results (depicted in Table 7-10, Table 7-11, and Figure 7-10) show that treated groups reviewed a higher percentage of the design document than the control groups (for both group and individual projects) which was shown to be statistically significant. The results indicate that the use of process patterns had a positive effect on the percentage of the design document reviewed. The results show that the design document review was improved by a sensitivity margin of 38.1% in group projects and 32.1% in individual projects. Based on these results, it can therefore be deduced that the application of process patterns improves design reviews.</p> <p>It has been further shown that the mean difference between the treated groups and control groups in group projects, in terms of the percentage of reviewed design document, was significantly higher than in the individual projects. This indicates that the employment of process patterns has been more effective on group projects than on individual projects for this metric. It can therefore be deduced that process patterns are more effective on group projects than individual projects in improving design reviews.</p>

Project type	Experimental Type	Semester	Mean	N (no. of cases)	Std. Deviation
Group-Project	Treated	Semester 1	6.13	47	2.393
		Semester 2	5.87	18	2.278
	Control	Semester 1	9.39	47	1.888
		Semester 2	9.76	20	2.451
Individual-Project	Treated	Semester 2	7.30	66	2.556
	Control	Semester 2	9.46	62	2.148

Table 7-12 Statistics for the no. of methods per class



The Boxplot depicts the range of values for 'Method per Class' metric for both treated and control groups for different project types and in different semesters. The boxplot indicates that the range of values and median in all the groupings is higher for the control groups than for the treated groups.

Figure 7-11 Boxplot for No. of Methods per Class

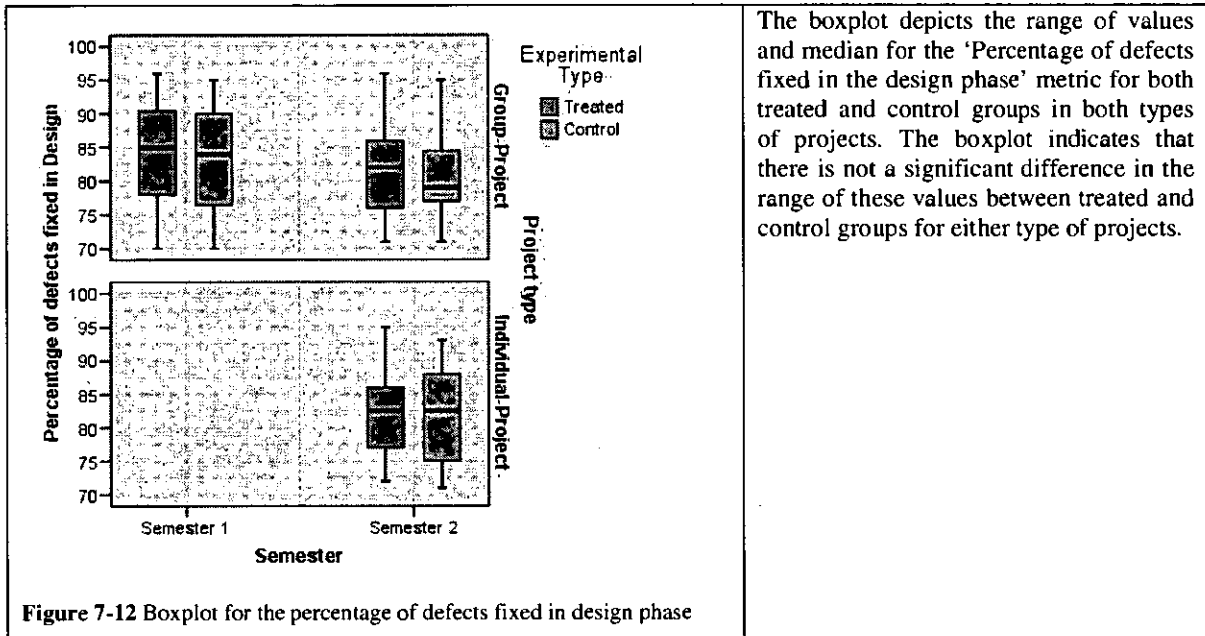
No. of Methods per Class (Methods per Class Ratio)			
Statistical significance analysis of mean difference between treated and control groups			
Operation	Independent Variables	Dependent Variable	Projects
2 × 2 independent measure ANOVA	Experiment group-type Semester	Methods per Class Ratio	Group Projects
Significance p-value	Description	Sensitivity/Margin (%)	
0.007	There was a main effect for experiment group-type.	33.8	
0.789	The main effect for the semester was not significant		
0.193	The main effect for the interaction was not significant		
Operation	Independent Variable	Dependent Variable	Projects
Independent Samples t-test	Experiment group-type	Methods per Class Ratio	Individual Projects
Significance p-value	Description	Sensitivity/Margin (%)	
0.021	There was a main effect for experiment group-type	20.3	
<b>Conclusion</b>	There is a statistically significant difference between the treated and control groups in both individual and group projects for this metric. This indicates that the treatment (i.e. use of process patterns) was statistically effective.		
Statistical significance analysis of the effect of project type on the experiment group-types			
Operation	Independent Variables	Dependent Variable	Projects
2 × 2 independent measure ANOVA	Experiment group-type Project-type	Methods per Class Ratio	Group Projects Individual projects
Significance p-value	Description		
0.003	There is a main effect for experiment group-type		
0.031	The main effect for the group-type is significant		
0.013	The main effect for the interaction is significant		
<b>Conclusion</b>	There is a statistically significant difference between group and individual projects in terms of the mean difference between the treated and control groups for this metric. This indicates that the treatment (i.e. use of process patterns) was statistically more effective on group projects than on individual projects for this metric.		
Correlation Analysis for the 'no. of logins' and 'this metric' for group and individual projects			
Correlation Coef. (r)	Significance (P)	Description	Project type
0.421	0.009	There is a statistically significant positive correlation	Group Projects
0.311	0.010	There is a statistically significant positive correlation	Individual Projects

Table 7-13 Statistical significance analysis for the 'No. of methods per class' metric

<b>Metric Result Summary</b>	
No. of Methods per Class (Methods per Class Ratio)	
<p>The results (depicted in Table 7-12, Table 7-13, and Figure 7-11) show that treated groups produced a lower number of methods per class than the control groups for both group and individual projects, which was shown to be statistically significant. Classes with lower number of methods are less complex and more maintainable and reusable. The results show that the method per class ratio was improved by a sensitivity margin of 33.8% in group projects and 20.3% in individual projects. The results, therefore, indicate that the use of process patterns has a significantly positive effect in improving the modularity and granularity of the design. Based on these results, it can therefore be deduced that the application of process patterns improves the design modularity and granularity.</p>	
<p>It has been further shown that the difference between the treated and control groups in group projects, in terms of the number of methods per class, was significantly higher than in the individual projects. This indicates that the employment of process patterns has been more effective on group projects than on individual projects for this metric. It can therefore be deduced that process patterns are more effective on group projects than individual projects in improving the design modularity and granularity.</p>	

Project type	Experimental Type	Semester	Mean	N (no. of cases)	Std. Deviation
Group-Project	Treated	Semester 1	83.91	47	7.967
		Semester 2	81.72	18	7.094
	Control	Semester 1	82.91	47	8.035
		Semester 2	81.00	20	6.593
Individual-Project	Treated	Semester 2	82.26	66	6.388
	Control	Semester 2	81.76	62	6.963

Table 7-14 Statistics for the percentage of defects fixed in the design phase



The boxplot depicts the range of values and median for the 'Percentage of defects fixed in the design phase' metric for both treated and control groups in both types of projects. The boxplot indicates that there is not a significant difference in the range of these values between treated and control groups for either type of projects.

Figure 7-12 Boxplot for the percentage of defects fixed in design phase

Percentage of Defects Fixed (Design phase)			
Statistical significance analysis of mean difference between treated and control groups			
Operation	Independent Variables	Dependent Variable	Projects
2 × 2 independent measure ANOVA	Experiment group-type Semester	Percentage of defects fixed (Design phase)	Group Projects
Significance p-value	Description	Sensitivity/Margin (%)	
0.302	The main effect for experiment group-type is not significant	N/A	
0.712	The main effect for the semester is not significant	(No significant difference between treated and control groups)	
0.603	The main effect for the interaction is not significant		
Operation	Independent Variable	Dependent Variable	Projects
Independent Samples t-test	Experiment group-type	Percentage of defects fixed (Design phase)	Individual Projects
Significance p-value	Description	Sensitivity/Margin (%)	
0.119	The main effect for experiment group-type is not significant	N/A	
(No significant difference between treated and control groups)			
<b>Conclusion</b>	There is not a statistically significant difference between the treated and control groups in either individual or group projects for this metric. This indicates that treatment (i.e. use of process patterns) was not statistically effective.		
Statistical significance analysis of the effect of project type on the experiment group-types			
Operation	Independent Variables	Dependent Variable	Projects
2 × 2 independent measure ANOVA	Experiment group-type Project-type	Percentage of defects fixed (Design phase)	Group Projects Individual projects
Significance p-value	Description		
0.163	There is a main effect for experiment group-type		
0.543	The main effect for the group-type is not significant		
0.363	The main effect for the interaction is not significant		
<b>Conclusion</b>	There is not a statistically significant difference between group and individual projects in terms of the mean difference between the treated and control groups for this metric. This indicates that treatment (i.e. use of process patterns) was not statistically more effective on either group or individual projects for this metric.		
Correlation Analysis for the 'no. of logins' and 'this metric' for group and individual projects			
Correlation Coef. (r)	Significance (P)	Description	Project type
0.113	0.130	There is not a statistically significant positive correlation	Group Projects
0.084	0.182	There is not a statistically significant positive correlation	Individual Projects

Table 7-15 Statistical significance analysis for the 'percentage of defects fixed' metric

<i>Metric Result Summary</i>	
Percentage of Defects Fixed (Design phase)	
<p>The results (depicted in Table 7-14, Table 7-15, and Figure 7-12) show that there is not a statistically significant mean difference between the treated and control groups for both group and individual projects in terms of the percentage of defects fixed in the Design phase. Based on these results it can therefore be concluded that the application of process patterns do not significantly increase the proportion of the defects fixed in the Design phase.</p>	
<p>It has been further shown that the difference between the treated groups and control in group projects in terms of the percentage of defects fixed in the Design phase was not significantly different to those in the individual projects. It can therefore be deduced that process patterns are not more effective in group projects than individual projects for this metric.</p>	

Project type	Experimental Type	Semester	Mean	N (no. of cases)	Std. Deviation
Group-Project	Treated	Semester 1	7.04	47	1.503
		Semester 2	7.44	18	1.756
	Control	Semester 1	6.76	47	1.449
		Semester 2	7.15	20	1.309
Individual-Project	Treated	Semester 2	9.65	66	1.060
	Control	Semester 2	8.98	62	0.864

Table 7-16 Statistics for the percentage of the Design phase time spent in testing

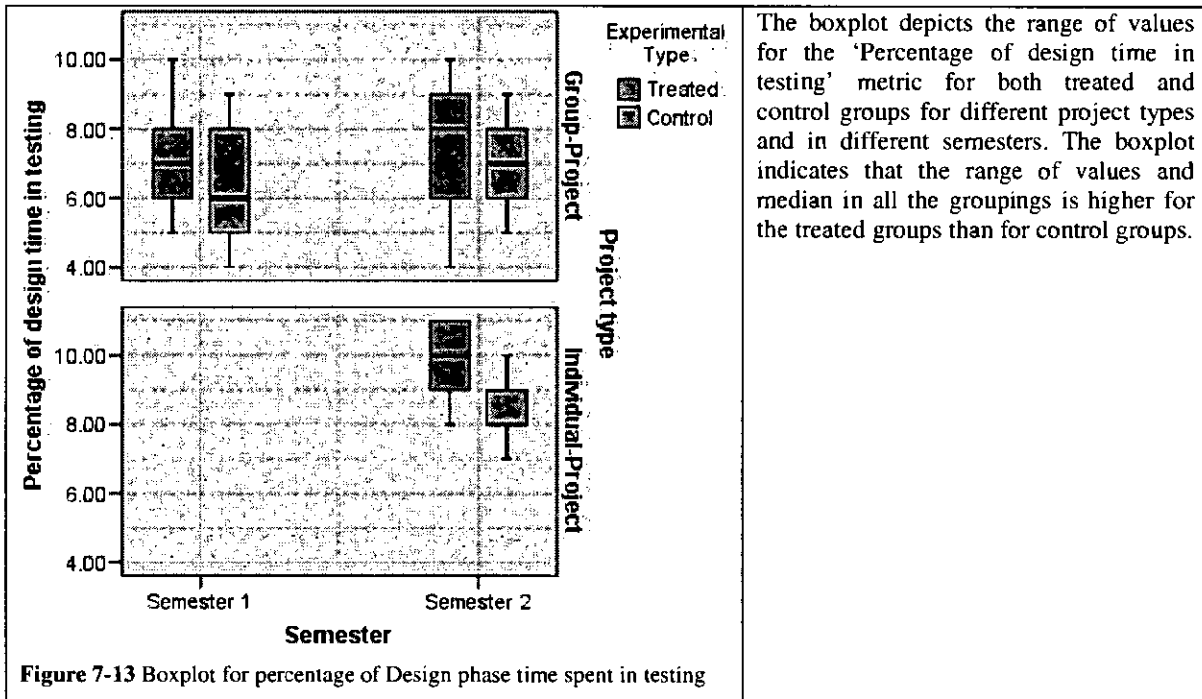


Figure 7-13 Boxplot for percentage of Design phase time spent in testing

Percentage of Phase Time Spent in Testing (Design phase)			
Statistical significance analysis of mean difference between treated and control groups			
Operation	Independent Variables	Dependent Variable	Projects
2 x 2 independent measure ANOVA	Experiment group-type Semester	Percentage of phase time spent in testing (Design phase)	Group Projects
Significance p-value	Description	Sensitivity/Margin (%)	
0.251	The main effect for the experiment group-type is not significant	N/A	
0.422	The main effect for the semester is not significant	(No significant difference between treated and control groups)	
0.314	The main effect for the interaction is not significant		
Operation	Independent Variable	Dependent Variable	Projects
Independent Samples t-test	Experiment group-type	Percentage of phase time spent in testing (Design phase)	Individual Projects
Significance p-value	Comment	Sensitivity/Margin (%)	
0.121	There main effect for experiment group-type is not significant	N/A (No significant difference between treated and control groups)	
<b>Conclusion</b>	There is not a statistically significant difference between the treated and control groups in either individual or group projects for this metric. This indicates that treatment (i.e. use of process patterns) was not statistically effective.		
Statistical significance analysis of the effect of project type on the experiment group-types			
Operation	Independent Variables	Dependent Variable	Projects
2 x 2 independent measure ANOVA	Experiment group-type Project-type	Percentage of Phase Time Spent in Testing (Design phase)	Group Projects Individual projects
Significance p-value	Comment		
0.213	The main effect for the experiment group-type is not significant		
0.421	The main effect for the group-type is not significant		
0.311	The main effect for the interaction is not significant		
<b>Conclusion</b>	There is not a statistically significant difference between group and individual projects in terms of the mean difference between the treated and control groups for this metric. This indicates that treatment (i.e. use of process patterns) was not statistically more effective on either group or individual projects for this metric.		
Correlation Analysis for the 'no. of logins' and 'this metric' for group and individual projects			
Correlation Coef. (r)	Significance (P)	Description	Project type
0.120	0.203	There is not a statistically significant positive correlation	Group Projects
0.091	0.190	There is not a statistically significant positive correlation	Individual Projects

Table 7-17 Statistical significance analysis for the 'percentage of Design phase time spent in testing' metric

Metric Result Summary
Percentage of Phase Time Spent in Testing (Design)
The results (depicted in Table 7-16, Table 7-17, and Figure 7-13) show that the percentage of the Design phase time spent for testing was not significantly different between the treated and control groups for both group and individual projects. Based on these results, it can be deduced that the application of process patterns do not significantly affect the proportion of the Design phase time spent in testing.
It has been further shown that the difference between the treated groups and control in group projects in terms of the percentage of the Design phase time spent in testing was not significantly different to those in the individual projects. It can therefore be deduced that process patterns are not more effective in group projects than individual projects for this metric.

### 7.6.3 Implementation Phase

In this section the result of the third development phase (i.e. Implementation) are presented. The following metrics are analysed:



- Comment density
- Percentage of source code reviewed
- Percentage of defects fixed
- Defect density
- Productivity (Implementation phase)
- Productivity (Overall)
- Percentage of Implementation phase time spent in testing

Project type	Experimental Type	Semester	Meen	N (no. of cases)	Std. Deviation
Group-Project	Treated	Semester 1	7.57	47	1.802
		Semester 2	6.11	18	1.745
	Control	Semester 1	4.59	47	1.233
		Semester 2	4.35	20	0.939
Individual-Project	Treated	Semester 2	6.84	66	1.501
	Control	Semester 2	4.84	62	1.414

Table 7-18 Statistics for the Comment Density

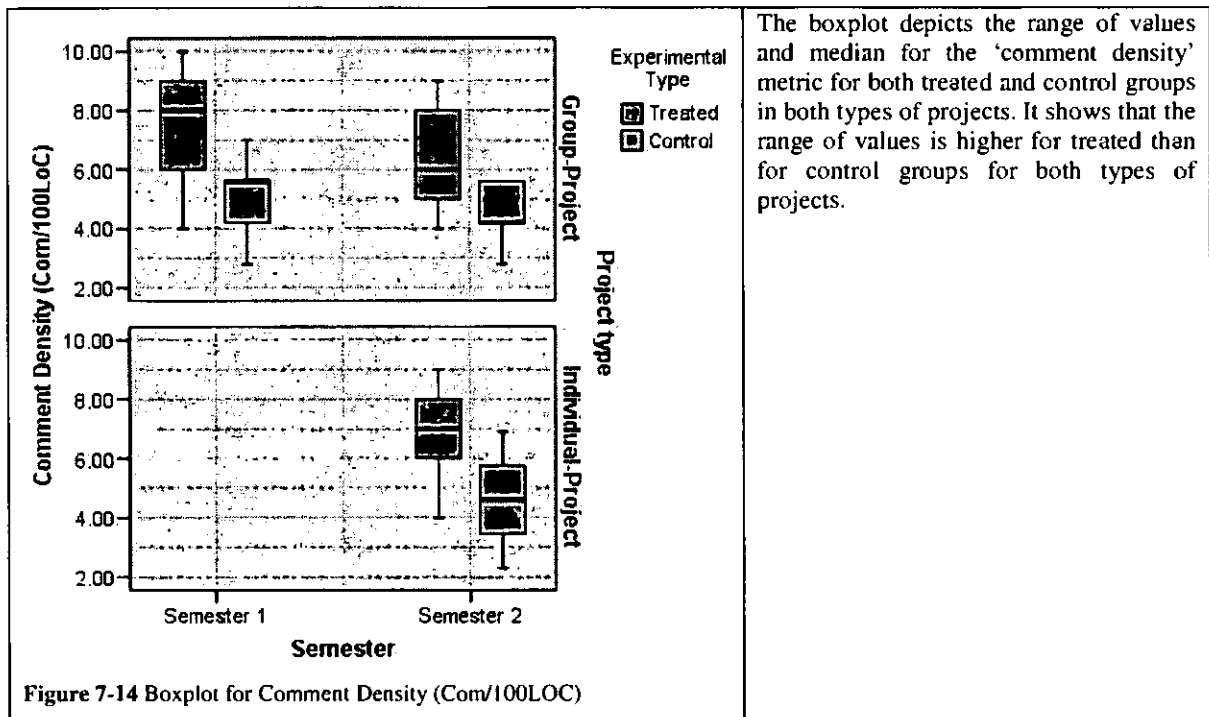


Figure 7-14 Boxplot for Comment Density (Com/100LOC)

Comment Density			
Statistical significance analysis of mean difference between treated and control groups			
Operation	Independent Variables	Dependent Variable	Projects
2 × 2 independent measure ANOVA	Experiment group-type Semester	Comment Density	Group Projects
Significance p-value	Description		Sensitivity margin (%)
0.000	There is a main effect for experiment group-type		48.8
0.883	The main effect for the semester is not significant		
0.763	The main effect for the interaction is not significant		
Operation	Independent Variable	Dependent Variable	Projects
Independent Samples t-test	Experiment group-type	Comment Density	Individual Projects
Significance p-value	Description		Sensitivity margin (%)
0.001	There is a main effect for experiment group-type		37.6
<b>Conclusion</b>	There is a statistically significant difference between the treated and control groups in both individual and group projects for this metric. This indicates that the treatment (i.e. use of process patterns) was statistically effective.		
Statistical significance analysis of the effect of project type on the experiment group-types			
Operation	Independent Variables	Dependent Variable	Projects
2 × 2 independent measure ANOVA	Experiment group-type Project-type	Comment Density	Group Projects Individual projects
Significance p-value	Description		
0.023	There is a main effect for experiment group-type		
0.031	The main effect for the group-type is significant		
0.016	The main effect for the interaction is significant		
<b>Conclusion</b>	There is a statistically significant difference between group and individual projects in terms of the mean difference between the treated and control groups for this metric. This indicates that the treatment (i.e. use of process patterns) was statistically more effective on group projects than on individual projects for this metric.		
Correlation Analysis for the 'no. of logins' and 'this metric' for group and individual projects			
Correlation Coef.(r)	Significance (P)	Description	Project type
0.481	0.008	There is a statistically significant positive correlation	Group Projects
0.376	0.009	There is a statistically significant positive correlation	Individual Projects

Table 7-19 Statistical significance analysis for the 'Comment density' metric

<i>Metric Result Summary</i>	
Comment Density	
<p>The results (depicted in Table 7-18, Table 7-19, and Figure 7-14) show that the comment density value was higher for treated groups than control groups for both group and individual projects, which is shown to be statistically significant. The results show that the comment density was improved by a sensitivity margin of 48.8% in group projects and 37.6% in individual projects. The results, therefore, indicate that the use of process patterns has a significantly positive effect in increasing the comment density. Based on these results, it can therefore be deduced that the application of process patterns improves comment density in the produced source code.</p>	
<p>It has been further shown that the mean difference between the treated groups and control in the group projects, in terms of the comment density, was statistically significantly higher than in the individual projects. This indicates that the employment of process patterns has been more effective on group projects than on individual projects for this metric. It can therefore be deduced that process patterns are more effective in group projects than in individual projects in improving comment density.</p>	

Project type	Experimental Type	Semester	Mean	N (no. of cases)	Std. Deviation
Group-Project	Treated	Semester 1	16.15	47	3.169
		Semester 2	15.72	18	2.674
	Control	Semester 1	11.24	47	1.982
		Semester 2	9.82	20	2.074
Individual-Project	Treated	Semester 2	13.51	66	2.381
	Control	Semester 2	9.28	62	2.085

Table 7-20 Statistics for the percentage of source code reviewed

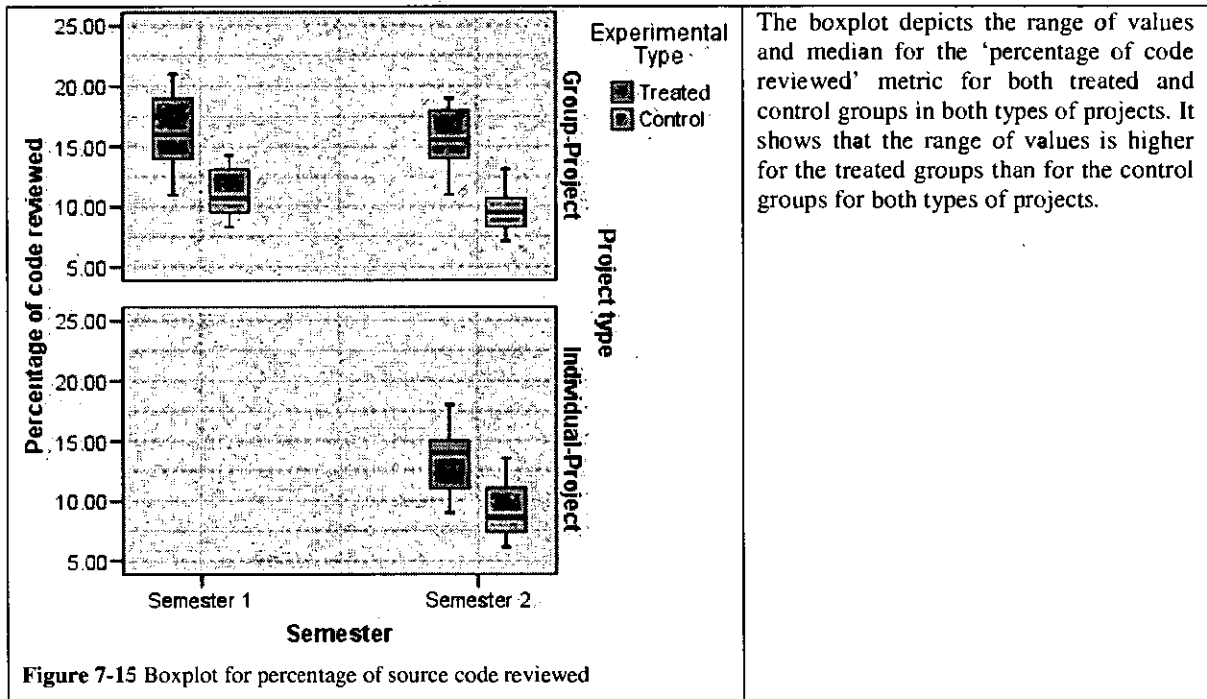


Figure 7-15 Boxplot for percentage of source code reviewed

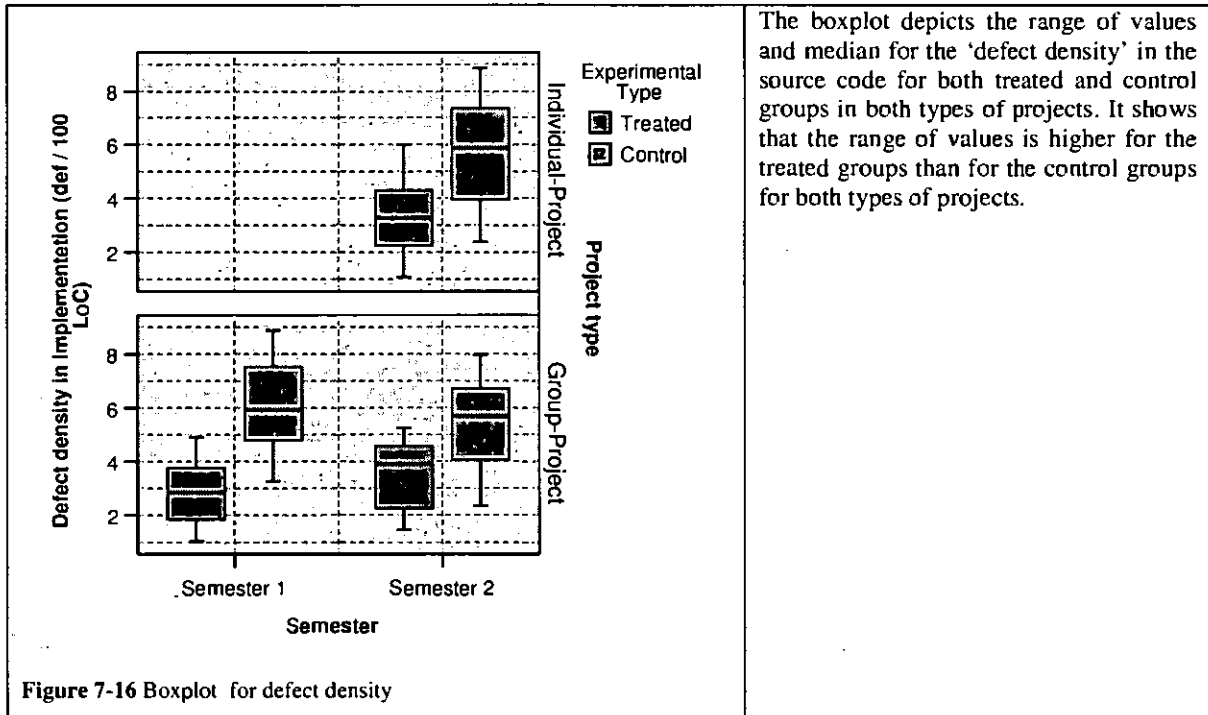
Percentage of Source Code Reviewed			
Statistical significance analysis of mean difference between treated and control groups			
Operation	Independent Variables	Dependent Variable	Projects
2 × 2 independent measure ANOVA	Experiment group-type Semester	Percentage of Source Code Reviewed	Group Projects
Significance p-value	Description	Sensitivity Margin (%)	
0.002	There is a main effect for experiment group-type	47.5	
0.183	The main effect for the semester is not significant		
0.323	The main effect for the interaction is not significant		
Operation	Independent Variable	Dependent Variable	Projects
Independent Samples t-test	Experiment group-type	Percentage of Source Code Reviewed	Individual Projects
Significance p-value	Description	Sensitivity Margin (%)	
0.000	There is a main effect for experiment group-type	41.9	
<b>Conclusion</b>	There is a statistically significant difference between the treated and control groups in both individual and group projects for this metric. This indicates that the treatment (i.e. use of process patterns) was statistically effective.		
Statistical significance analysis of the effect of project type on the experiment group-types			
Operation	Independent Variables	Dependent Variable	Projects
2 × 2 independent measure ANOVA	Experiment group-type Project-type	Percentage of Source Code Reviewed	Group Projects Individual projects
Significance p-value	Description		
0.013	There is a main effect for experiment group-type		
0.021	The main effect for the group-type is significant		
0.012	The main effect for the interaction is significant		
<b>Conclusion</b>	There is a statistically significant difference between group and individual projects in terms of the mean difference between the treated and control groups for this metric. This indicates that the treatment (i.e. use of process patterns) was statistically more effective on group projects than on individual projects for this metric.		
Correlation Analysis for the 'no. of logins' and 'this metric' for group and individual projects			
Correlation Coef. (r)	Significance (P)	Description	Project type
0.289	0.021	There is a statistically significant positive correlation	Group Projects
0.321	0.003	There is a statistically significant positive correlation	Individual Projects

Table 7-21 Statistical significance analysis for the 'percentage of source code reviewed' metric

<b>Metric Result Summary</b>	
<b>Percentage of source code reviewed</b>	
<p>The results (depicted in Table 7-20, Table 7-21, and Figure 7-15) show that treated groups reviewed a higher percentage of the source code than the control groups in both group and individual projects, which was shown to be statistically significant. The results show that code review was improved by a sensitivity margin of 47.5% in group projects and 41.9% in individual projects. The higher the percentage of the source code reviewed the higher the likelihood of detecting and correcting any defects. The results, therefore, indicate that the use of process patterns has a significant positive effect in increasing the percentage of source code reviewed. Based on these results, it can therefore be deduced that the application of process patterns improves code reviews.</p>	
<p>It has been further shown that the mean difference between the treated groups and control in the group projects, in terms of the percentage of reviewed source code, was statistically significantly higher than in the individual projects. This indicates that the employment of process patterns has been more effective on group projects than on individual projects for this metric. It can therefore be deduced that process patterns are more effective in group projects than in individual projects in improving code review.</p>	

Project type	Experimental Type	Semester	Mean	N (no. of cases)	Std. Deviation
Group-Project	Treated	Semester 1	2.83	47	1.168
		Semester 2	3.58	18	1.294
	Control	Semester 1	6.10	47	1.708
		Semester 2	5.44	20	1.756
Individual-Project	Treated	Semester 2	3.36	66	1.381
	Control	Semester 2	5.72	62	1.919

Table 7-22 Statistics for the defect density in the source code



The boxplot depicts the range of values and median for the 'defect density' in the source code for both treated and control groups in both types of projects. It shows that the range of values is higher for the treated groups than for the control groups for both types of projects.

Figure 7-16 Boxplot for defect density

Defect Density			
Statistical significance analysis of mean difference between treated and control groups			
Operation	Independent Variables	Dependent Variable	Projects
2 × 2 independent measure ANOVA	Experiment group-type Semester	Defect Density	Group Projects
Significance p-value	Description	Sensitivity Margin (%)	
0.002	There is a main effect for experiment group-type	39.2	
0.781	The main effect for the semester is not significant		
0.293	The main effect for the interaction is not significant		
Operation	Independent Variable	Dependent Variable	Projects
Independent Samples t-test	Experiment group-type	Defect Density	Individual Projects
Significance p-value	Description	Sensitivity Margin (%)	
0.000	There is a main effect for experiment group-type	37.3	
<b>Conclusion</b>	There is a statistically significant difference between the treated and control groups in both individual and group projects for this metric. This indicates that the treatment (i.e. use of process patterns) was statistically effective.		
Statistical significance analysis of the effect of project type on the experiment group-types			
Operation	Independent Variables	Dependent Variable	Projects
2 × 2 independent measure ANOVA	Experiment group-type Project-type	Defect Density	Group Projects Individual projects
Significance p-value	Description		
0.103	There is not a main effect for experiment group-type		
0.553	The main effect for the group-type is not significant		
0.363	The main effect for the interaction is not significant		
<b>Conclusion</b>	There is not a statistically significant difference between group and individual projects in terms of the mean difference between the treated and control groups for this metric. This indicates that treatment (i.e. use of process patterns) was not statistically more effective on either group or individual projects for this metric.		
Correlation Analysis for the 'no. of logins' and 'this metric' for group and individual projects			
Correlation Coef. (r)	Significance (P)	Description	Project type
0.294	0.036	There is a statistically significant positive correlation	Group Projects
0.327	0.012	There is a statistically significant positive correlation	Individual Projects

Table 7-23 Statistical significance analysis for the 'defect density' metric

Metric Result Summary	
Defect density	
<p>The results (depicted in Table 7-22, Table 7-23, and Figure 7-16) show that treated groups had a lower defect density in the source code than the control groups for both group and individual projects which was shown to be statistically significant. The lower the defect density, the higher the quality of code. The results show that defect density was improved by a sensitivity margin of 39.2% in group projects and 37.3% in individual projects. The results, therefore, indicate that the use of process patterns has a significant positive effect in lowering the defect density in the source code. Based on these results, it can therefore be deduced that the application of process patterns lowers the defect density in the source code.</p>	
<p>It has been further shown that the difference between the treated groups and control in group projects in terms of the defect density was not significantly different to those in the individual projects. It can therefore be deduced that process patterns are not more effective on group projects than individual projects for this metric.</p>	

Project type	Experimental Type	Semester	Mean	N (no. of cases)	Std. Deviation
Group-Project	Treated	Semester 1	8.98	47	1.319
		Semester 2	9.07	18	0.914
	Control	Semester 1	6.61	47	0.904
		Semester 2	6.63	20	1.194
Individual-Project	Treated	Semester 2	7.55	66	1.274
	Control	Semester 2	5.99	62	0.974

Table 7-24 Statistics for productivity in the Implementation phase

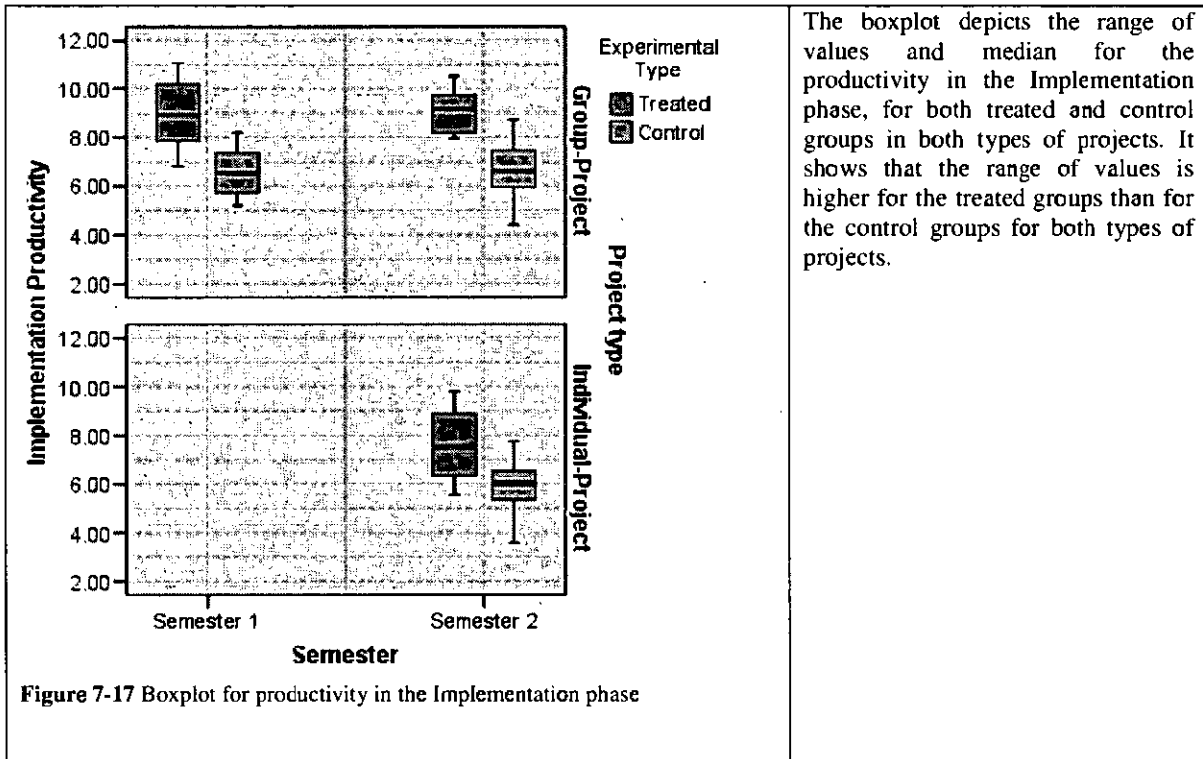


Figure 7-17 Boxplot for productivity in the Implementation phase

Implementation Productivity			
Statistical significance analysis of mean difference between treated and control groups			
Operation	Independent Variables	Dependent Variable	Projects
2 × 2 independent measure ANOVA	Experiment group-type Semester	Productivity (Implementation phase)	Group Projects
Significance p-value	Description	Sensitivity Margin (%)	
0.001	There is a main effect for experiment group-type	33.3	
0.283	The main effect for the semester is not significant		
0.122	The main effect for the interaction is not significant		
Operation	Independent Variable	Dependent Variable	Projects
Independent Samples t-test	Experiment group-type	Productivity (Implementation phase)	Individual Projects
Significance p-value	Description	Sensitivity Margin (%)	
0.002	There is a main effect for experiment group-type	23.8	
<b>Conclusion</b>	There is a statistically significant difference between the treated and control groups in both individual and group projects for this metric. This indicates that the treatment (i.e. use of process patterns) was statistically effective.		
Statistical significance analysis of the effect of project type on the experiment group-types			
Operation	Independent Variables	Dependent Variable	Projects
2 × 2 independent measure ANOVA	Experiment group-type Project-type	Productivity (Implementation phase)	Group Projects Individual projects
Significance p-value	Description		
0.014	There is a main effect for experiment group-type		
0.031	The main effect for the group-type is significant		
0.019	The main effect for the interaction is significant		
<b>Conclusion</b>	There is a statistically significant difference between group and individual projects in terms of the mean difference between the treated and control groups for this metric. This indicates that the treatment (i.e. use of process patterns) was statistically more effective on group projects than on individual projects for this metric.		
Correlation Analysis for the 'no. of logins' and 'this metric' for group and individual projects			
Correlation Coef. (r)	Significance (P)	Description	Project type
0.291	0.013	There is a statistically significant positive correlation	Group Projects
0.402	0.000	There is a statistically significant positive correlation	Individual Projects

Table 7-25 Statistical significance analysis for the 'Implementation productivity' metric

Metric Result Summary
<p><b>Implementation Productivity</b></p> <p>The results (depicted in Table 7-24, Table 7-25, and Figure 7-17) showed that productivity in the Implementation phase was higher in treated groups than in control groups in both group and individual projects, which was shown to be statistically significant. For the group projects, the productivity in the Implementation phase was found to be 9.0 LOC per hour (72 LOC per day) for treated groups and 6.6 LOC per hour (52.8 LOC per day) in the case of control groups. For the individual projects, the productivity in the Implementation phase was found to be 7.5 LOC per hour (60 LOC per day) for the treated groups and 6.0 LOC per hour (48 LOC per day) for the control groups. The results show that the implementation productivity was improved by a sensitivity margin of 33.3% in group projects and 23.8% in individual projects. The results, therefore, indicate that the use of process patterns has a significant positive effect in increasing productivity in the Implementation phase. Based on these results, it can therefore be deduced that the application of process patterns improves implementation productivity.</p> <p>It has been further shown that the mean difference between the treated groups and control in the group projects, in terms of productivity, was statistically significantly higher than in the individual projects. This indicates that the employment of process patterns has been more effective on group projects than on individual projects for this metric. It can therefore be deduced that process patterns are more effective in group projects than in individual projects in increasing productivity in the Implementation phase.</p>



Project type	Experimental Type	Semester	Mean	N (no. of cases)	Std. Deviation
Group-Project	Treated	Semester 1	3.52	47	0.522
		Semester 2	3.44	18	0.358
	Control	Semester 1	2.70	47	0.361
		Semester 2	2.65	20	0.477
Individual-Project	Treated	Semester 2	2.91	66	0.500
	Control	Semester 2	2.40	62	0.402

Table 7-26 Statistics for the overall productivity

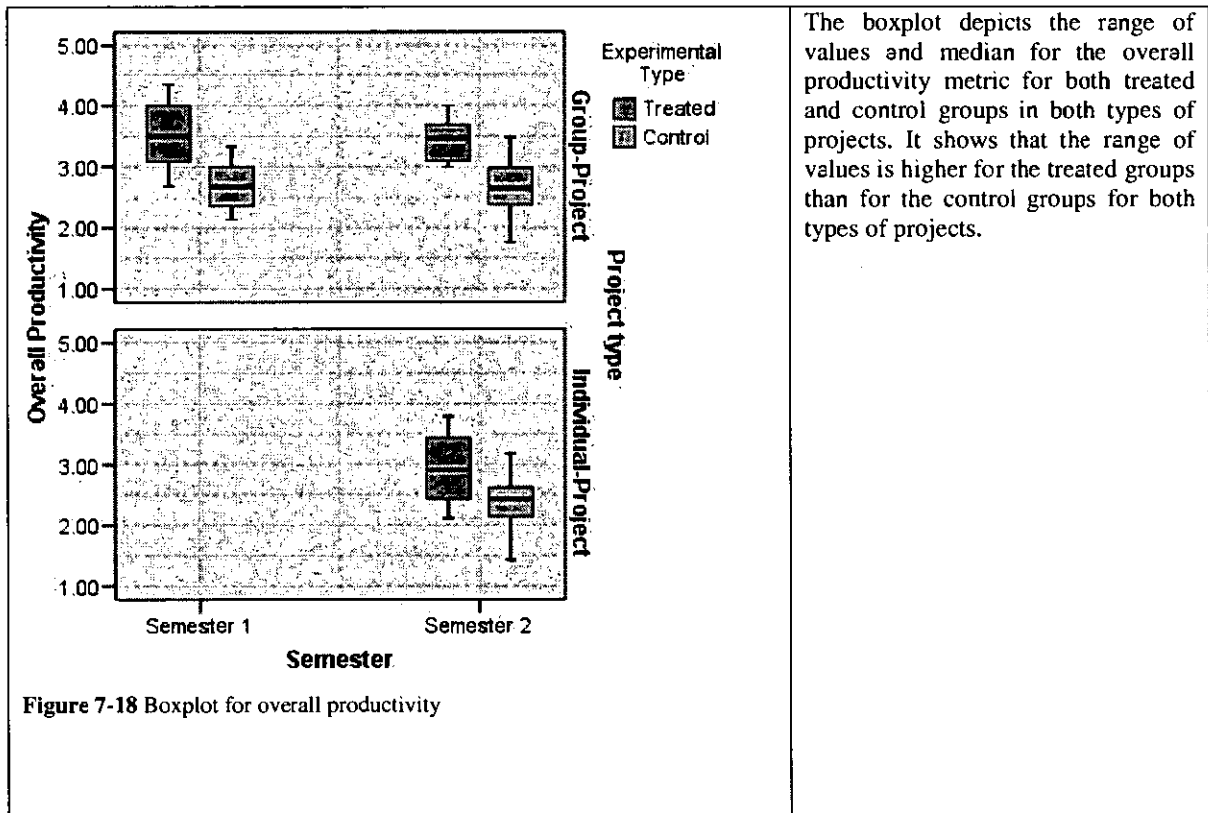


Figure 7-18 Boxplot for overall productivity

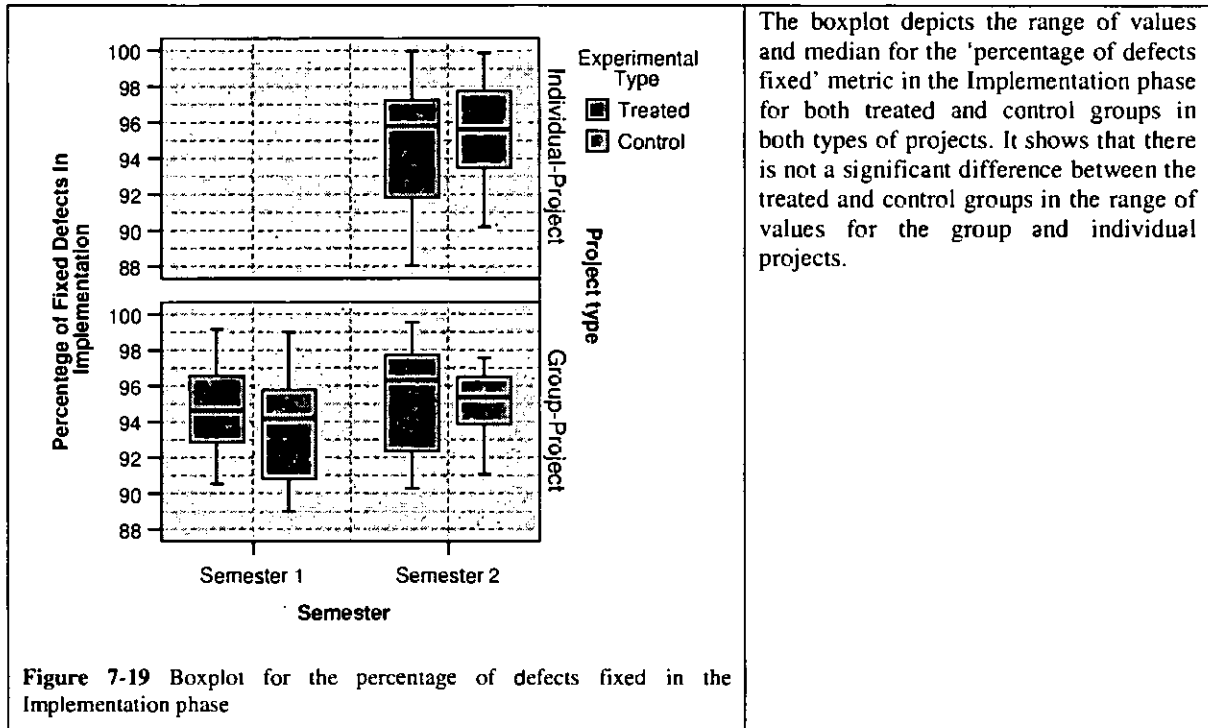
Overall Productivity			
Statistical significance analysis of mean difference between treated and control groups			
Operation	Independent Variables	Dependent Variable	Projects
2 × 2 independent measure ANOVA	Experiment group-type Semester	Overall Productivity	Group Projects
Significance p-value	Description	Sensitivity/Margin (%)	
0.003	There is a main effect for experiment group-type	27.1	
0.328	The main effect for the semester is not significant		
0.153	The main effect for the interaction is not significant		
Operation	Independent Variable	Dependent Variable	Projects
Independent Samples t-test	Experiment group-type	Overall Productivity	Individual Projects
Significance p-value	Description	Sensitivity/Margin (%)	
0.001	There is a main effect for experiment group-type	19.2	
<b>Conclusion</b>	There is a statistically significant difference between the treated and control groups in both individual and group projects for this metric. This indicates that the treatment (i.e. use of process patterns) was statistically effective.		
Statistical significance analysis of the effect of project type on the experiment group-types			
Operation	Independent Variables	Dependent Variable	Projects
2 × 2 independent measure ANOVA	Experiment group-type Project-type	Overall Productivity	Group Projects Individual projects
Significance p-value	Description		
0.015	There is a main effect for experiment group-type		
0.22	The main effect for the group-type is significant		
0.014	The main effect for the interaction is significant		
<b>Conclusion</b>	There is a statistically significant difference between group and individual projects in terms of the mean difference between the treated and control groups for this metric. This indicates that the treatment (i.e. use of process patterns) was statistically more effective on group projects than on individual projects for this metric.		
Correlation Analysis for the 'no. of logins' and 'this metric' for group and individual projects			
Correlation Coef. (r)	Significance (P)	Description	Project type
0.325	0.010	There is a statistically significant positive correlation	Group Projects
0.296	0.028	There is a statistically significant positive correlation	Individual Projects

Table 7-27 Statistical significance analysis for the 'overall productivity' metric

<b>Metric Result Summary</b>
Overall Productivity
The results (depicted in Table 7-26, Table 7-27, and Figure 7-18) showed that the overall productivity (i.e. complete development project) was higher in treated groups than in control groups in both group and individual projects, which was shown to be statistically significant. For the group projects, the overall productivity (i.e. complete development project) was found to be 3.5 LOC per hour (28 per day) for treated groups and 2.7 LOC per hour (21.6 LOC per day) in the case of control groups. For the individual projects, the overall productivity was found to be 2.9 LOC per hour (23.2 LOC per day) for treated groups and 2.4 LOC per hour (19.2 LOC per day) for the control groups. The results show that overall productivity was improved by a sensitivity margin of 27.1% in group projects and 19.2% in individual projects. The results, therefore, indicate that the use of process patterns has a significant positive effect in increasing the overall productivity. Based on these results, it can therefore be deduced that the application of process patterns improves productivity.
It has been further shown that the mean difference between the treated groups and control in the group projects, in terms of the overall productivity, was statistically significantly higher than in the individual projects. This indicates that the employment of process patterns has been more effective on group projects than on individual projects for this metric. It can therefore be deduced that process patterns are more effective in group projects than in individual projects in increasing productivity.

Project type	Experimental Type	Semester	Mean	N (no. of cases)	Std. Deviation
Group-Project	Treated	Semester 1	94.75	47	2.504
		Semester 2	95.25	18	3.109
	Control	Semester 1	93.78	47	3.002
		Semester 2	95.02	20	2.027
Individual-Project	Treated	Semester 2	94.88	66	3.507
	Control	Semester 2	95.55	62	2.538

Table 7-28 Statistics for the percentage of defects fixed in the Implementation phase



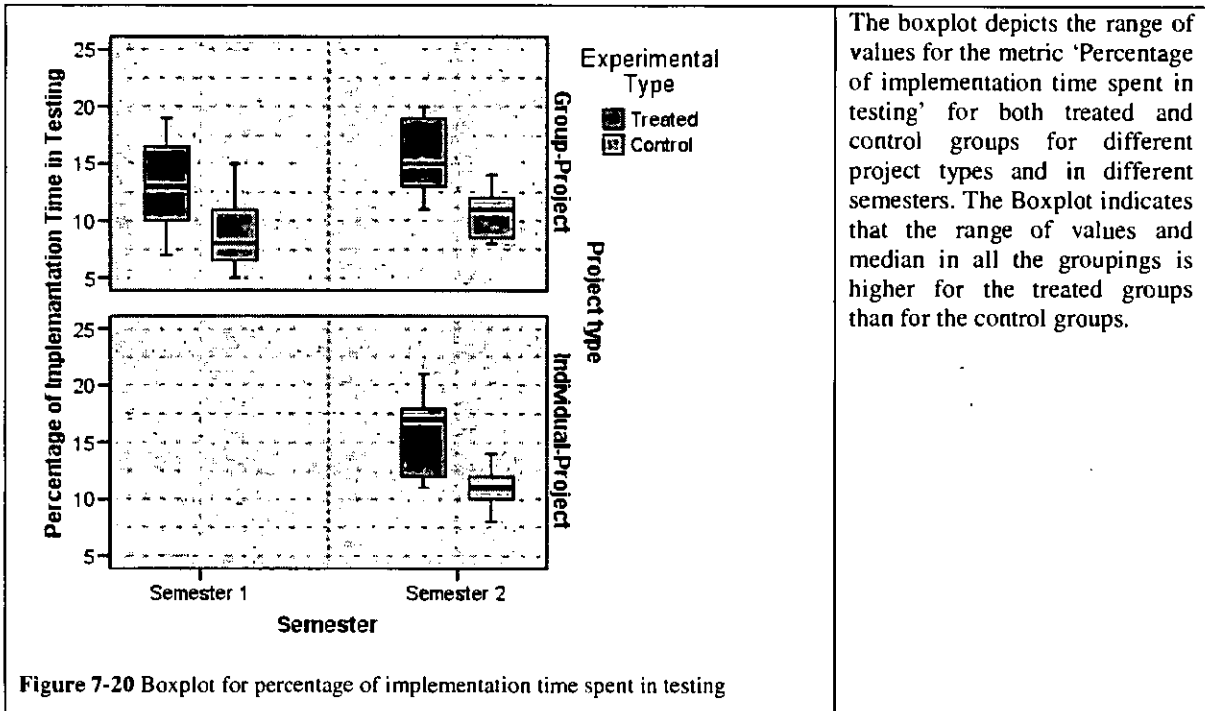
Percentage of defects fixed (Implementation phase)			
Statistical significance analysis of mean difference between treated and control groups			
Operation	Independent Variables	Dependent Variable	Projects
2 × 2 independent measure ANOVA	Experiment group-type Semester	Percentage of defects fixed (Implementation phase)	Group Projects
Significance p-value	Description		Sensitivity/Margin (%)
0.212	The main effect for experiment group-type is not significant		N/A
0.863	The main effect for the semester is not significant		(No significant difference between treated and control groups)
0.730	The main effect for the interaction is not significant		
Operation	Independent Variable	Dependent Variable	Projects
Independent Samples t-test	Experiment group-type	Percentage of defects fixed (Implementation phase)	Individual Projects
Significance p-value	Description		Sensitivity/Margin (%)
0.097	The main effect for experiment group-type is not significant		N/A (No significant difference between treated and control groups)
<b>Conclusion</b>	There is not a statistically significant difference between the treated and control groups in either individual or group projects for this metric. This indicates that treatment (i.e. use of process patterns) was not statistically effective.		
Statistical significance analysis of the effect of project type on the experiment group-types			
Operation	Independent Variables	Dependent Variable	Projects
2 × 2 independent measure ANOVA	Experiment group-type Project-type	Percentage of defects fixed (Implementation phase)	Group projects Individual projects
Significance p-value	Description		
0.122	There is a main effect for experiment group-type		
0.582	The main effect for the group-type is not significant		
0.313	The main effect for the interaction is not significant		
<b>Conclusion</b>	This indicates that treatment (i.e. application of process patterns) was not statistically more effective on group projects than on individual projects.		
Correlation Analysis for the 'no. of logins' and 'this metric' for group and individual projects			
Correlation Coef. (r)	Significance (P)	Description	Project type
0.095	0.191	There is not a statistically significant positive correlation	Group Projects
0.086	0.210	There is not a statistically significant positive correlation	Individual Projects

Table 7-29 Statistical significance analysis for the 'percentage of defects fixed' metric

<b>Metric/Result/Summary</b>
<p align="center"><b>Percentage of defects fixed (Implementation phase)</b></p> <p>The results (depicted in Table 7-28, Table 7-29, and Figure 7-19) show that there is not a statistically significant mean difference between the treated and control groups for both group and individual projects in terms of the percentage of defects fixed in the Implementation phase. Based on the results it can therefore be concluded that the application of process patterns does not improve the quality of the Implementation phase in terms of the percentage of defects fixed in the Implementation phase.</p> <p>It has been further shown that the difference between the treated groups and control in group projects in terms of the percentage of defects fixed in the Implementation was not significantly different to those in the individual projects. It can therefore be deduced that process patterns are not more effective in group projects than individual projects for this metric.</p>

Project type	Experimental Type	Semester	Mean	N (no. of cases)	Std. Deviation
Group-Project	Treated	Semester 1	13.17	47	3.571
		Semester 2	15.56	18	2.935
	Control	Semester 1	8.89	47	2.838
		Semester 2	10.60	20	2.010
Individual-Project	Treated	Semester 2	15.85	66	3.119
	Control	Semester 2	11.05	62	1.683

Table 7-30 Statistics for the percentage of Implementation phase time spent in testing



The boxplot depicts the range of values for the metric 'Percentage of implementation time spent in testing' for both treated and control groups for different project types and in different semesters. The Boxplot indicates that the range of values and median in all the groupings is higher for the treated groups than for the control groups.

Figure 7-20 Boxplot for percentage of implementation time spent in testing

Percentage of Phase Time Spent in Testing (Implementation phase)			
Statistical significance analysis of mean difference between treated and control groups			
Operation	Independent Variables	Dependent Variable	Projects
2 × 2 independent measure ANOVA	Experiment group-type Semester	Percentage of phase time spent in testing (Implementation phase)	Group Projects
Significance p-value	Description	Sensitivity Margin (%)	
0.041	There is a main effect for experiment group-type	41.2	
0.743	The main effect for the semester is not significant		
0.345	The main effect for the interaction is not significant		
Operation	Independent Variable	Dependent Variable	Projects
Independent Samples t-test	Experiment group-type	Percentage of phase time spent in testing (Implementation phase)	Individual Projects
Significance p-value	Description	Sensitivity Margin (%)	
0.022	There is a main effect for experiment group-type	38.4	
<b>Conclusion</b>	There is a statistically significant difference between the treated and control groups in both individual and group projects for this metric. This indicates that the treatment (i.e. use of process patterns) was statistically effective.		
Statistical significance analysis of the effect of project type on the experiment group-types			
Operation	Independent Variables	Dependent Variable	Projects
2 × 2 independent measure ANOVA	Experiment group-type Project-type	Percentage of phase time spent in testing (Implementation phase)	Group Projects Individual projects
Significance p-value	Description		
0.031	There is a main effect for experiment group-type		
0.020	The main effect for the group-type is significant		
0.031	The main effect for the interaction is significant		
<b>Conclusion</b>	There is a statistically significant difference between group and individual projects in terms of the mean difference between the treated and control groups for this metric. This indicates that the treatment (i.e. use of process patterns) was statistically more effective on group projects than on individual projects for this metric.		
Correlation Analysis for the 'no. of logins' and 'this metric' for group and individual projects			
Correlation Coef.(r)	Significance (P)	Description	Project type
0.074	0.194	There is not a statistically significant positive correlation	Group Projects
0.103	0.096	There is not a statistically significant positive correlation	Individual Projects

Table 7-31 Results of significance analysis for the 'percentage of phase time spent in testing' metric

Metric/Result/Summary
<p align="center"><b>Percentage of Phase time Spent in Testing (Implementation phase)</b></p> <p>The results (depicted in Table 7-30, Table 7-31, and Figure 7-20) show that there is a statistically significant difference between the treated and control groups for both project types (i.e. group and individual) in the percentage of the development time spent in the Implementation phase. The results show that the percentage of phase time spent in implementation was improved by a sensitivity margin of 41.2% in group projects and 38.2% in individual projects. The results, therefore, indicate that the use of process patterns has a significant positive effect in increasing the percentage of phase time spent in tests. Based on these results, it can therefore be deduced that the application of process patterns improves the proportion of phase time spent in tests</p> <p>It has been further shown that the statistically significant difference between the treated groups and control in group projects, in terms of the percentage of the Implementation phase time spent for testing, was higher than in the individual projects. This indicates that the employment of process patterns has been more effective on group projects than on individual projects for this metric. It can therefore be deduced that process patterns are more effective in group projects than in individual projects for this metric.</p>

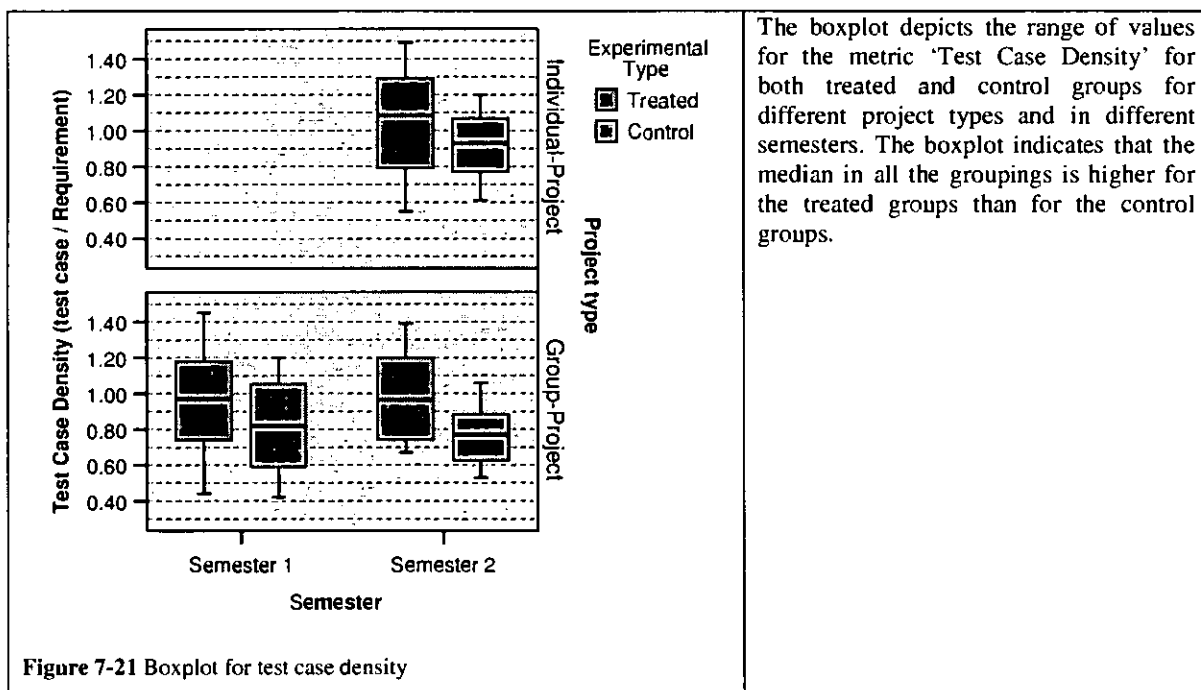
### 7.6.4 Delivery Phase

In this section the result of the final development phase (i.e. Delivery) are presented. The following metrics are analysed:

- Test Case Density
- Percentage of defects fixed
- Percentage of Delivery phase time spent in testing

Project type	Experimental Type	Semester	Mean	N (no. of cases)	Std. Deviation
Group-Project	Treated	Semester 1	0.96	47	0.281
		Semester 2	0.98	18	0.236
	Control	Semester 1	0.83	47	0.244
		Semester 2	0.77	20	0.161
Individual-Project	Treated	Semester 2	1.05	66	0.288
	Control	Semester 2	0.92	62	0.185

Table 7-32 Statistics for test case density in the Delivery phase



The boxplot depicts the range of values for the metric 'Test Case Density' for both treated and control groups for different project types and in different semesters. The boxplot indicates that the median in all the groupings is higher for the treated groups than for the control groups.

Figure 7-21 Boxplot for test case density

Test Case Density			
Statistical significance analysis of mean difference between treated and control groups			
Operation	Independent Variables	Dependent Variable	Projects
2 × 2 independent measure ANOVA	Experiment group-type Semester	Test Case Density	Group Projects
Significance p-value	Description		Sensitivity margin (%)
0.011	There is a main effect for <i>experiment group-type</i>		19.4
0.713	The main effect for the <i>semester</i> is not significant		
0.319	The main effect for the interaction is not significant		
Operation	Independent Variable	Dependent Variable	Projects
Independent Samples t-test	Experiment group-type	Test Case Density	Individual Projects
Significance p-value	Description		Sensitivity margin (%)
0.020	There is a main effect for <i>experiment group-type</i>		13.0
<b>Conclusion</b>	There is a statistically significant difference between the treated and control groups in both individual and group projects for this metric. This indicates that the treatment (i.e. use of process patterns) was statistically effective.		
Statistical significance analysis of the effect of project type on the experiment group-types			
Operation	Independent Variables	Dependent Variable	Projects
2 × 2 independent measure ANOVA	Experiment group-type Project-type	Test Case Density	Group Projects Individual projects
Significance p-value	Description		
0.013	There is a main effect for <i>experiment group-type</i>		
0.021	The main effect for the <i>group-type</i> is significant		
0.010	The main effect for the interaction is significant		
<b>Conclusion</b>	There is a statistically significant difference between group and individual projects in terms of the mean difference between the treated and control groups for this metric. This indicates that the treatment (i.e. use of process patterns) was statistically more effective on group projects than on individual projects for this metric.		
Correlation Analysis for the 'no. of logins' and 'this metric' for group and individual projects			
Correlation Coef. (r)	Significance (P)	Description	Project type
0.391	0.009	There is a statistically significant positive correlation	Group Projects
0.412	0.001	There is a statistically significant positive correlation	Individual Projects

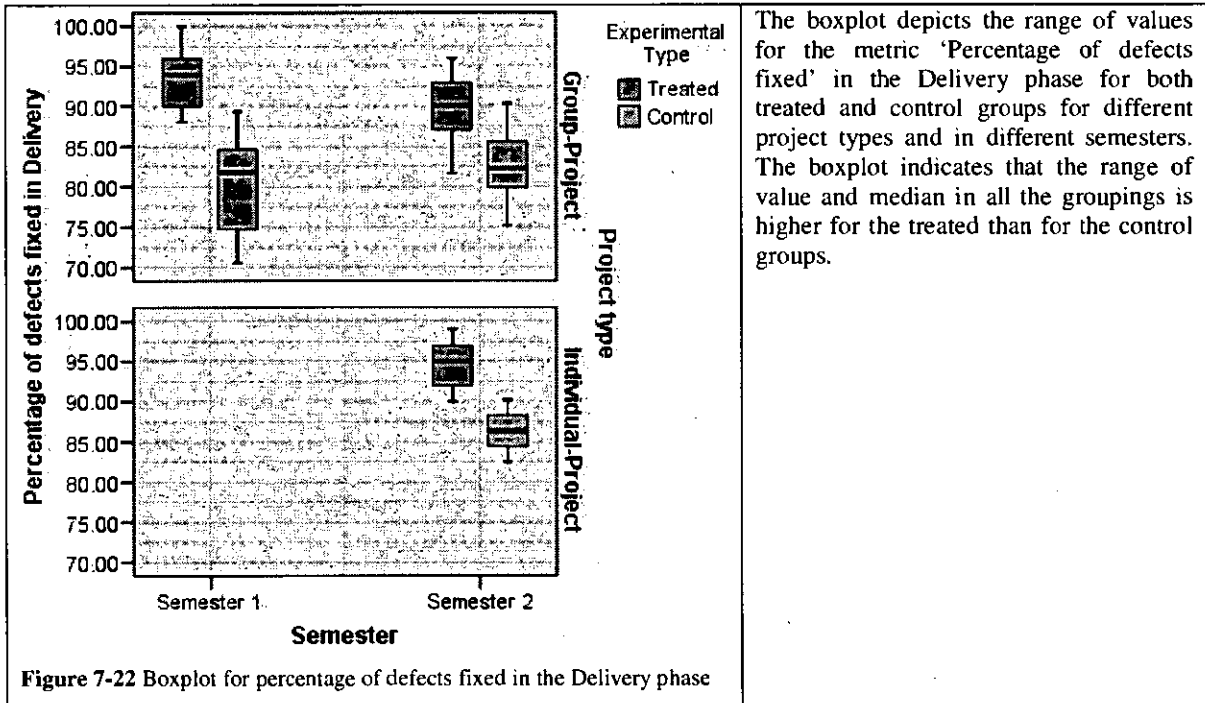
Table 7-33 Statistical significance analysis for the 'test case density' metric

<b>Metric Result Summary</b>
<b>Test Case density (Delivery phase)</b>
The results (depicted in Table 7-32, Table 7-33, and Figure 7-21) show that the test case density was higher for the treated groups than the control groups in both group and individual projects, which were shown to be statistically significant. The results show that the test case density was improved by a sensitivity margin of 19.4% in group projects and 13.0% in individual projects. The results, therefore, indicate that the use of process patterns has a significant positive effect in increasing the test case density. Based on these results, it can therefore be deduced that the application of process patterns improves the test case density.
It has been further shown that the mean difference between the treated groups and control in the group projects, in terms of the test case density, was statistically significantly higher than in the individual projects. This indicates that the employment of process patterns has been more effective on group projects than on individual projects for this metric. It can therefore be deduced that process patterns are more effective in group projects than in individual projects in improving test case density.



Project type	Experimental Type	Semester	Mean	N (no. of cases)	Std. Deviation
Group-Project	Treated	Semester 1	93.23	47	3.577
		Semester 2	89.92	18	4.000
	Control	Semester 1	80.42	47	5.889
		Semester 2	82.53	20	4.394
Individual-Project	Treated	Semester 2	94.61	66	2.505
	Control	Semester 2	86.31	62	2.327

Table 7-34 Statistics for the percentage of defects fixed in the Delivery phase



The boxplot depicts the range of values for the metric 'Percentage of defects fixed' in the Delivery phase for both treated and control groups for different project types and in different semesters. The boxplot indicates that the range of value and median in all the groupings is higher for the treated than for the control groups.

Figure 7-22 Boxplot for percentage of defects fixed in the Delivery phase

Percentage of Defects Fixed (Delivery phase)			
Statistical significance analysis of mean difference between treated and control groups			
Operation	Independent Variables	Dependent Variable	Projects
2 × 2 independent measure ANOVA	Experiment group-type Semester	Percentage of defects fixed	Group Projects
Significance p-value	Description	Sensitivity Margin (%)	
0.022	There is a main effect for experiment group-type	11.5	
0.793	The main effect for the semester is not significant		
0.249	The main effect for the interaction is not significant		
Operation	Independent Variable	Dependent Variable	Projects
Independent Samples t-test	Experiment group-type	Percentage of defects fixed	Individual Projects
Significance p-value	Description	Sensitivity Margin (%)	
0.030	There is a main effect for experiment group-type	8.8	
<b>Conclusion</b>	There is a statistically significant difference between the treated and control groups in both individual and group projects for this metric. This indicates that the treatment (i.e. use of process patterns) was statistically effective.		
Statistical significance analysis of the effect of project type on the experiment group-types			
Operation	Independent Variables	Dependent Variable	Projects
2 × 2 independent measure ANOVA	Experiment group-type Project-type	Percentage of defects fixed	Group Projects Individual projects
Significance p-value	Description		
0.130	The main effect for experiment group-type is not significant		
0.873	The main effect for the group-type is not significant		
0.303	The main effect for the interaction is not significant		
<b>Conclusion</b>	This indicates that treatment (i.e. application of process patterns) was not statistically more effective on group projects than on individual projects.		
Correlation Analysis for the 'no. of logins' and 'this metric' for group and individual projects			
Correlation Coef. (r)	Significance (P)	Description	Project type
0.327	0.0131	There is a statistically significant positive correlation	Group Projects
0.297	0.023	There is a statistically significant positive correlation	Individual Projects

Table 7-35 Statistical significance analysis for the 'percentage of defects fixed' metric

Metric Result Summary	
Percentage of defects fixed (Delivery phase)	
<p>The results (depicted in Table 7-34, Table 7-35, and Figure 7-22) show that there is a statistically significant mean difference between the treated and control groups for both group and individual projects in terms of the percentage of the defects fixed in the Delivery phase. The results show that requirement traceability was improved by a sensitivity margin of 11.5% in group projects and 8.8% in individual projects. The results, therefore, indicate that the use of process patterns has a significant positive effect in increasing the percentage of defects fixed. Based on the results it can therefore be concluded that the application of process patterns increases the proportion of defects fixed in the Delivery phase.</p>	
<p>It has been further shown that the mean difference between the treated groups and control in group projects in terms of the percentage of defects was not significantly different to those in the individual projects. It can therefore be deduced that process patterns are not more effective in group projects than individual projects for this metric.</p>	

Project type	Experimental Type	Semester	Mean	N (no. of cases)	Std. Deviation
Group-Project	Treated	Semester 1	79.68	47	7.599
		Semester 2	73.28	18	9.423
	Control	Semester 1	59.09	47	8.140
		Semester 2	57.25	20	10.804
Individual-Project	Treated	Semester 2	70.86	66	7.736
	Control	Semester 2	62.53	62	15.383

Table 7-36 Statistics for the percentage Delivery phase time spent in testing

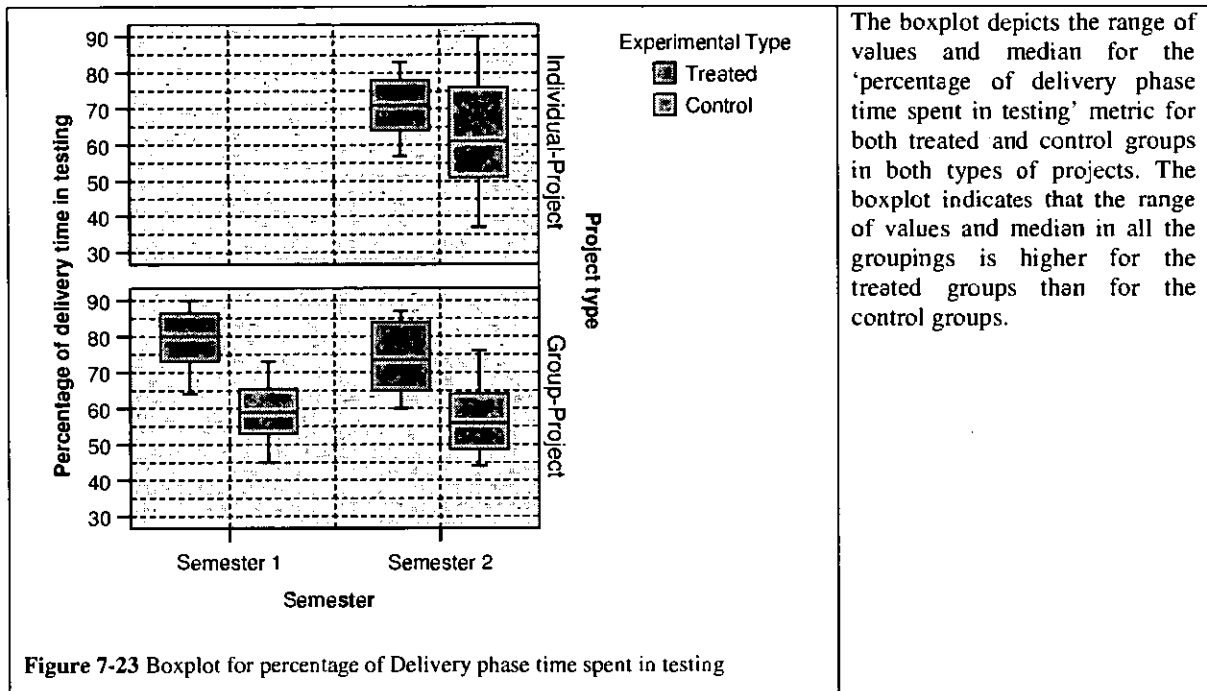


Figure 7-23 Boxplot for percentage of Delivery phase time spent in testing

Percentage of Phase Time Spent in Testing (Delivery phase)			
Statistical significance analysis of mean difference between treated and control groups			
Operation	Independent Variables	Dependent Variable	Projects
2 x 2 independent measure ANOVA	Experiment group-type Semester	Percentage of phase time spent in testing (Delivery phase)	Group Projects
Significance p-value	Description		Sensitivity Margin (%)
0.000	There is a main effect for experiment group-type		29.0
0.394	The main effect for the semester is not significant		
0.440	The main effect for the interaction is not significant		
Operation	Independent Variable	Dependent Variable	Projects
Independent Samples t-test	Experiment group-type	Percentage of phase time spent in testing (Delivery phase)	Individual Projects
Significance p-value	Description		Sensitivity Margin (%)
0.016	There is a main effect for experiment group-type		12.1
<b>Conclusion</b> There is a statistically significant difference between the treated and control groups in both individual and group projects for this metric. This indicates that the treatment (i.e. use of process patterns) was statistically effective.			
Statistical significance analysis of the effect of project type on the experiment group-types			
Operation	Independent Variables	Dependent Variable	Projects
2 x 2 independent measure ANOVA	Experiment group-type Project-type	Percentage of phase time spent in testing (Delivery phase)	Group Projects Individual projects
Significance p-value	Description		
0.000	There is a main effect for experiment group-type		
0.010	The main effect for the group-type is significant		
0.011	The main effect for the interaction is significant		
<b>Conclusion</b> There is a statistically significant difference between group and individual projects in terms of the mean difference between the treated and control groups for this metric. This indicates that the treatment (i.e. use of process patterns) was statistically more effective on group projects than on individual projects for this metric.			
Correlation Analysis for the 'no. of logins' and 'this metric' for group and individual projects			
Correlation Coef. (r)	Significance (P)	Description	Project type
0.487	0.002	There is a statistically significant positive correlation	Group Projects
0.319	0.006	There is a statistically significant positive correlation	Individual Projects

Table 7-37 Statistical analysis for the 'percentage of phase time spent in testing' metric

<b>Metric Result Summary</b>	
<b>Percentage of Phase Spent in Testing (Delivery phase)</b>	
<p>The results (depicted in Table 7-36, Table 7-37, and Figure 7-23) indicated that the percentage of Delivery phase time spent on testing was significantly higher for control groups than control for both group and individual projects. The results show that requirement traceability was improved by a sensitivity margin of 29.0% in group projects and 12.1% in individual projects. The results, therefore, indicate that the use of process patterns has a significant positive effect in increasing the test case density. Based on the results it can be deduced that the application of process patterns increased the proportion of the Delivery phase time spent in testing.</p>	
<p>It has been further shown that the mean difference between the treated groups and control in the group projects, in terms of the proportion of the Delivery phase time spent in testing, was statistically significantly higher than in the individual projects. This indicates that the employment of process patterns has been more effective on group projects than on individual projects for this metric. It can therefore be deduced that process patterns are more effective in group projects for this metric.</p>	

Apart from data collected through conducting a measurement process (presented above), the official marks awarded to projects by tutors for a number of attributes were also used in the experiment, which are presented and analysed in the following section.

## 7.7 Tutor Marks Results

As discussed in the research method chapter (Chapter 5), the experiment was conducted on real and official student project, which were marked by tutors after their completion. There were a number of project attributes, which were marked separately by tutors. There were four attributes marked (i.e. 'design and analysis', product, evaluation, and project management) which were directly related to the software attributes that the study was investigating. The attributes marked are depicted in Table 7-38 in relation to their respective development phases.

Officially Marked Attributes	Development Phases
Design and Analysis	Requirement analysis, Design
Evaluation	Delivery
Product	Delivery, Implementation
Project Management	Requirement Analysis, Design, Implementation, Delivery

Table 7-38 Relationships between the development phases marked attributes

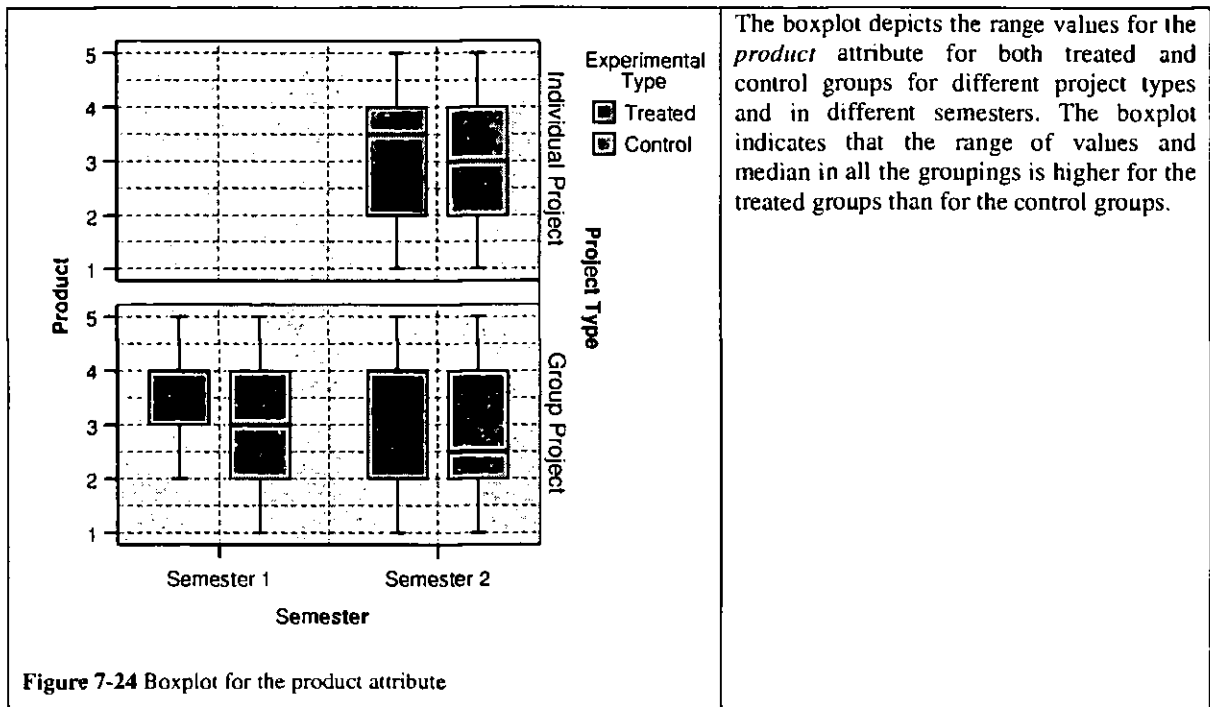
In this section, the statistical analysis of the marks awarded to the project by tutors for the four attributes stated above are presented.

### 7.7.1 Product

This section analyses the marks awarded by the tutors to the software *product* attribute of both group and individual projects. The product attribute, as evaluated and marked by the tutors, represents a general evaluation of the delivered software.

Project type	Experimental Type	Semester	Mean	N (no. of cases)	Std. Deviation
Group-Project	Treated	Semester 1	3.53	47	1.100
		Semester 2	3.44	18	1.247
	Control	Semester 1	2.98	47	0.989
		Semester 2	3.00	20	1.257
Individual-Project	Treated	Semester 2	3.29	66	1.262
	Control	Semester 2	2.87	62	1.337

Table 7-39 Statistics for the product attribute



The boxplot depicts the range values for the *product* attribute for both treated and control groups for different project types and in different semesters. The boxplot indicates that the range of values and median in all the groupings is higher for the treated groups than for the control groups.

Figure 7-24 Boxplot for the product attribute

Product			
Statistical significance analysis of mean difference between treated and control groups			
Operation	Independent Variables	Dependent Variable	Projects
2 × 2 independent measure ANOVA	Experiment group-type Semester	Product	Group Projects
Significance p-value	Description		Sensitivity/Margin (%)
0.004	There is a main effect for experiment group-type		15.2
0.123	The main effect for the semester is not significant		
0.343	The main effect for the interaction is not significant		
Operation	Independent Variable	Dependent Variable	Projects
Independent Samples t-test	Experiment group-type	Product	Individual Projects
Significance p-value	Description		Sensitivity/Margin (%)
0.001	There is a main effect for experiment group-type		12.9
<b>Conclusion</b> There is a statistically significant difference between the treated and control groups in both individual and group projects for this attribute. This indicates that the treatment (i.e. use of process patterns) was statistically effective.			
Statistical significance analysis of the effect of project type on the experiment group-types			
Operation	Independent Variables	Dependent Variable	Projects
2 × 2 independent measure ANOVA	Experiment group-type Project-type	Product	Group Projects Individual projects
Significance p-value	Description		
0.006	There is a main effect for experiment group-type		
0.033	The main effect for the group-type is significant		
0.019	The main effect for the interaction is significant		
<b>Conclusion</b> There is a statistically significant difference between group and individual projects in terms of the mean difference between the treated and control groups for this attribute. This indicates that the treatment (i.e. use of process patterns) was statistically more effective on group projects than on individual projects for this attribute.			
Correlation Analysis for the 'no. of logins' and 'this metric' for group and individual projects			
Correlation Coef. (r)	Significance (P)	Description	Project type
0.424	0.001	There is a statistically significant positive correlation	Group Projects
0.394	0.002	There is a statistically significant positive correlation	Individual Projects

Table 7-40 Statistical significance analysis for the 'product' attribute

**Result Summary**

**Product**

The results (depicted in Table 7-39, Table 7-40, and Figure 7-24) show that there is a statistically significant difference between the treated and control groups for both project types (i.e. group and individual) for the value of 'product'. The results show that the product quality was improved by a sensitivity margin of 15.2% in group projects and 12.9% in individual projects. The results, therefore, indicate that the use of process patterns has a significant positive effect in increasing the value of the product attribute. Based on these results, it can therefore be deduced that the application of process patterns improves the overall quality of the developed software.

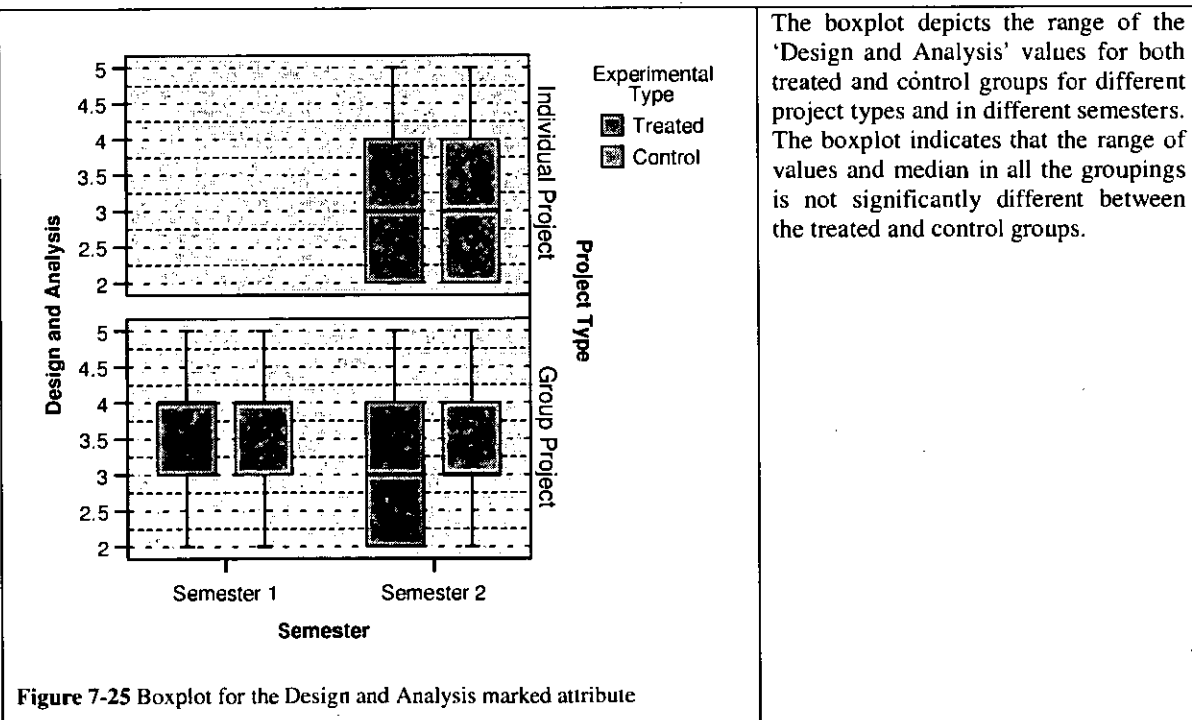
It has been further shown that the mean difference between the treated groups and control groups in group projects, in terms of the product, was statistically significantly higher than those in the individual projects. This indicates that the employment of process patterns has been more effective in group projects than in individual projects for this attribute. It can therefore be deduced that the process patterns are more effective on group projects than individual projects in improving the overall quality of the product.

**7.7.2 Design and Analysis**

This section analyse the marks awarded to the software *design and analysis* attribute.

Project Type	Experimental Type	Semester	Mean	N (no. of cases)	Std. Deviation
Group Project	Treated	Semester 1	3.64	47	0.819
		Semester 2	3.28	18	1.074
	Control	Semester 1	3.60	47	0.851
		Semester 2	3.20	20	0.768
Individual Project	Treated	Semester 2	3.12	66	0.985
	Control	Semester 2	3.06	62	0.956

**Table 7-41** Statistics for the design and analysis marked attribute



**Figure 7-25** Boxplot for the Design and Analysis marked attribute

Design and Analysis			
Statistical significance analysis of mean difference between treated and control groups			
Operation	Independent Variables	Dependent Variable	Projects
2 × 2 independent measure ANOVA	Experiment group-type Semester	Design and Analysis	Group Projects
Significance p-value	Description	Sensitivity Margin (%)	
0.202	There main effect for the <i>experiment group-type</i> is not significant	N/A	
0.703	The main effect for the <i>semester</i> is not significant		
0.623	The main effect for the interaction is not significant		
Operation	Independent Variable	Dependent Variable	Projects
Independent Samples t-test	Experiment group-type	Design and Analysis	Individual Projects
Significance p-value	Description	Sensitivity Margin (%)	
0.210	There is not a main effect for <i>experiment group-type</i>	N/A	
<b>Conclusion</b>	There is not a statistically significant difference between the treated and control groups in either individual or group projects for this attribute. This indicates that treatment (i.e. use of process patterns) was not statistically effective.		
Statistical significance analysis of the effect of project type on the <i>experiment group-types</i>			
Operation	Independent Variables	Dependent Variable	Projects
2 × 2 independent measure ANOVA	Experiment group-type Project-type	Design and Analysis	Group Projects Individual projects
Significance p-value	Description		
0.090	There is not a main effect for <i>experiment group-type</i>		
0.213	The main effect for the <i>group-type</i> is not significant		
0.381	The main effect for the interaction is not significant		
<b>Conclusion</b>	There is not a statistically significant difference between group and individual projects in terms of the mean difference between the treated and control groups for this attribute. This indicates that treatment (i.e. use of process patterns) was not statistically more effective on either group or individual projects for this metric.		
Correlation Analysis for the 'no. of logins' and 'this metric' for group and individual projects			
Correlation Coef. (r)	Significance (P)	Description	Project type
0.262	0.041	There is a statistically significant positive correlation	Group Projects
0.248	0.042	There is a statistically significant positive correlation	Individual Projects

Table 7-42 Statistical analysis for the 'design and analysis' marked attribute

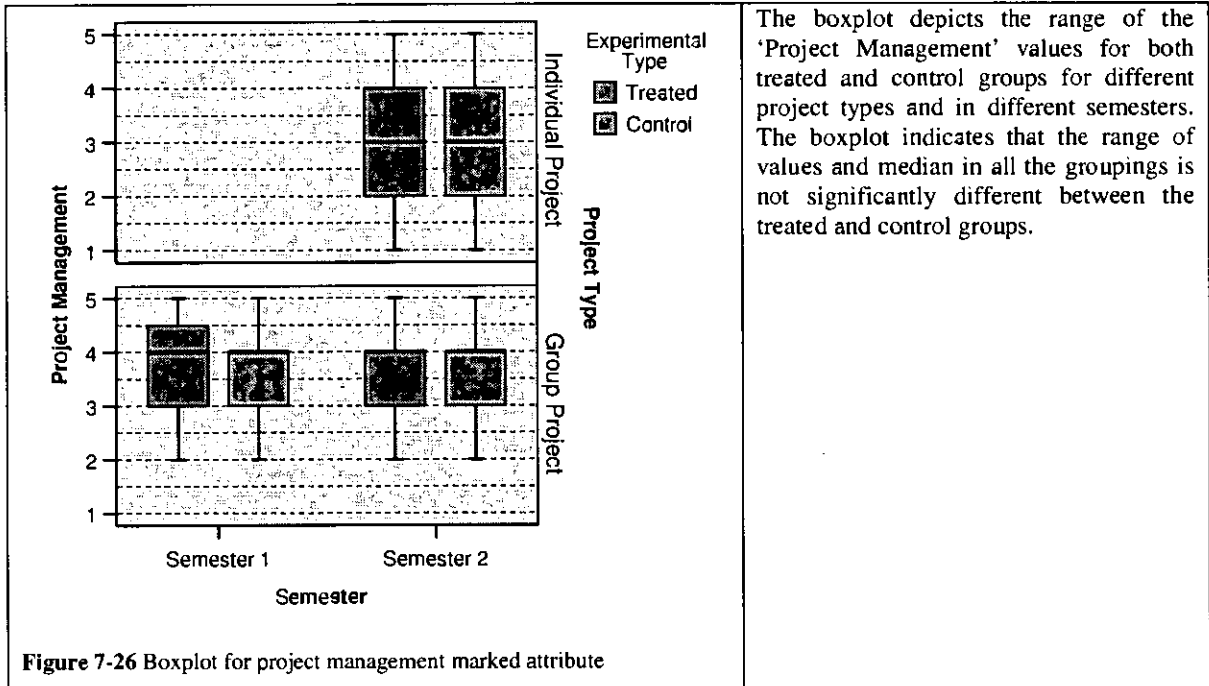
Result Summary	
<b>Design and Analysis</b>	
The results (depicted in Table 7-41, Table 7-42, and Figure 7-25) show that, while there is a slight difference between the control and treated groups in terms of the quality of the 'design and analysis' attribute, the difference is not statistically significant. This indicates that the use of process patterns did not have a significant effect on the design and analysis attribute of the project.	
The difference between the treated groups and control in group projects in terms of 'design and analysis' attribute was not significantly different to those in the individual projects. This indicates that the employment of process patterns was not more effective in either group or individual projects for this attribute.	

### 7.7.3 Project Management

This section analyse the marks awarded to the *project management* market attribute.

Project Type	Experimental Type	Semester	Mean	N (no. of cases)	Std. Deviation
Group Project	Treated	Semester 1	3.64	47	1.072
		Semester 2	3.67	18	0.840
	Control	Semester 1	3.57	47	0.927
		Semester 2	3.65	20	0.875
Individual Project	Treated	Semester 2	3.18	66	0.959
	Control	Semester 2	3.13	62	1.079

Table 7-43 Statistics for the *project management* marked attribute



The boxplot depicts the range of the 'Project Management' values for both treated and control groups for different project types and in different semesters. The boxplot indicates that the range of values and median in all the groupings is not significantly different between the treated and control groups.

Figure 7-26 Boxplot for project management marked attribute



Project Management			
Statistical significance analysis of mean difference between treated and control groups			
Operation	Independent Variables	Dependent Variable	Projects
2 × 2 independent measure ANOVA	Experiment group-type Semester	Project Management	Group Projects
Significance p-value	Description	Sensitivity Margin (%)	
0.232	There main effect for the <i>experiment group-type</i> is not significant	N/A	
0.783	The main effect for the <i>semester</i> is not significant		
0.513	The main effect for the interaction is not significant		
Operation	Independent Variable	Dependent Variable	Projects
Independent Samples t-test	Experiment group-type	Project Management	Individual Projects
Significance p-value	Description	Sensitivity Margin (%)	
0.110	There is not a main effect for <i>experiment group-type</i>	N/A	
<b>Conclusion</b>	There is not a statistically significant difference between the treated and control groups in either individual or group projects for this attribute. This indicates that treatment (i.e. use of process patterns) was not statistically effective.		
Statistical significance analysis of the effect of project type on the experiment group-types			
Operation	Independent Variables	Dependent Variable	Projects
2 × 2 independent measure ANOVA	Experiment group-type Project-type	Project Management	Group Projects Individual projects
Significance p-value	Description		
0.103	There is not a main effect for <i>experiment group-type</i>		
0.119	The main effect for the <i>group-type</i> is not significant		
0.317	The main effect for the interaction is not significant		
<b>Conclusion</b>	There is not a statistically significant difference between group and individual projects in terms of the mean difference between the treated and control groups for this attribute. This indicates that treatment (i.e. use of process patterns) was not statistically more effective on either group or individual projects for this metric.		
Correlation Analysis for the 'no. of logins' and 'this metric' for group and individual projects			
Correlation Coef. (r)	Significance (P)	Description	Project type
0.092	0.186	There is not a statistically significant positive correlation	Group Projects
0.051	0.217	There is not a statistically significant positive correlation	Individual Projects

Table 7-44 Statistical analysis for the 'Project Management' attribute

Result Summary	
Project Management	
<p>The results (depicted in Table 7-43, Table 7-44, and Figure 7-26) show that there is not a statistically significant difference between the control and treated groups in terms of project management as evaluated and marked by tutors. This indicates that the use of process patterns did not have a significant effect on the marked 'project management' attribute.</p>	
<p>The difference between the treated groups and control in group projects in terms project management was not significantly different to those in the individual projects. This indicates that the employment of process patterns was not more effective in group projects than individual projects for this attribute.</p>	

### 7.7.4 Evaluation

This section analyse the marks awarded to the software evaluation attribute.

Project Type	Experimental Type	Semester	Mean	N (no. of cases)	Std. Deviation
Group Project	Treated	Semester 1	3.23	47	0.890
		Semester 2	3.11	18	0.900
	Control	Semester 1	3.13	47	1.076
		Semester 2	3.05	20	0.686
Individual Project	Treated	Semester 2	3.18	66	0.959
	Control	Semester 2	3.13	62	1.079

Table 7-45 Statistics for the *evaluation* attribute

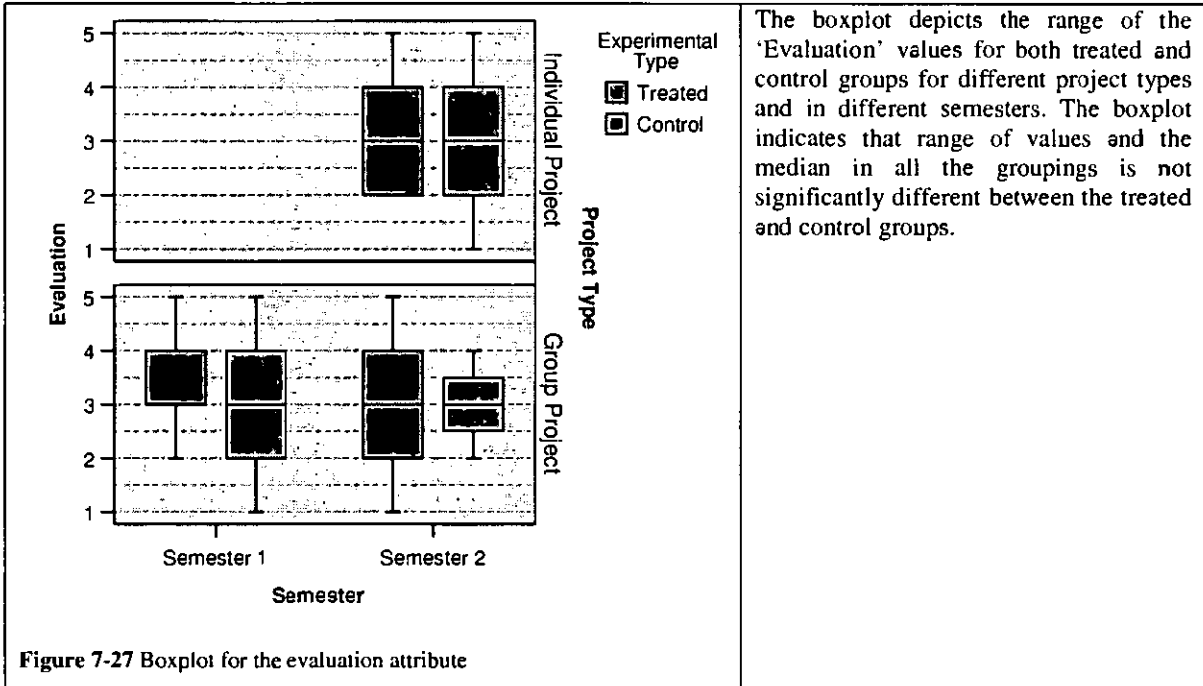


Figure 7-27 Boxplot for the evaluation attribute

Evaluation			
Statistical significance analysis of mean difference between treated and control groups			
Operation	Independent Variables	Dependent Variable	Projects
2 × 2 independent measure ANOVA	Experiment group-type Semester	Evaluation	Group Projects
Significance p-value	Description	Sensitivity/Margin (%)	
0.112	There main effect for the <i>experiment group-type</i> is not significant	N/A	
0.763	The main effect for the <i>semester</i> is not significant		
0.625	The main effect for the interaction is not significant		
Operation	Independent Variable	Dependent Variable	Projects
Independent Samples t-test	Experiment group-type	Evaluation	Individual Projects
Significance p-value	Description	Sensitivity/Margin (%)	
0.092	There is not a main effect for <i>experiment group-type</i>	N/A	
<b>Conclusion</b>	There is not a statistically significant difference between the treated and control groups in either individual or group projects for this attribute. This indicates that treatment (i.e. use of process patterns) was not statistically effective.		
Statistical significance analysis of the effect of project type on the experiment group-types			
Operation	Independent Variables	Dependent Variable	Projects
2 × 2 independent measure ANOVA	Experiment group-type Project-type	Evaluation	Group Projects Individual projects
Significance p-value	Description		
0.143	There is not a main effect for <i>experiment group-type</i>		
0.551	The main effect for the <i>group-type</i> is not significant		
0.363	The main effect for the interaction is not significant		
<b>Conclusion</b>	There is not a statistically significant difference between group and individual projects in terms of the mean difference between the treated and control groups for this attribute. This indicates that treatment (i.e. use of process patterns) was not statistically more effective on either group or individual projects for this metric.		
Correlation Analysis for the 'no. of logins' and 'this metric' for group and individual projects			
Correlation Coef.(r)	Significance (P)	Description	Project type
0.102	0.141	There is not a statistically significant positive correlation	Group Projects
0.068	0.197	There is not a statistically significant positive correlation	Individual Projects

Table 7-46 Statistical analysis for the *evaluation* attribute

Result/Summary	
Evaluation	
<p>The results (depicted in Table 7-45, Table 7-46, and Figure 7-27) show that there was a slight difference between the control and treated groups in terms of the quality of the evaluation. However, the difference was not statistically significant. This indicates that the use of process patterns did not have a significant effect on the 'evaluation' attribute.</p>	
<p>The difference between the treated groups and control in group projects in terms of the evaluation process was not significantly different to those in the individual projects. This indicates that the employment of process patterns was not more effective in either group or individual projects for this attribute.</p>	

The results of marks, awarded to the four attributes presented above, indicated that the application of process patterns improved the product attribute. The other three marked attributes were not significantly affected. This is further discussed in 8.5.1.

### 7.8 Subjects' Views on Process Pattern

The treated subjects were asked two 4-point Likert scale questions on their experience of using process patterns. These two questions are given below:

- How useful did you find process patterns in doing your project?

Not at all  Slightly  Moderately  Very

2. How difficult/easy did you find the process patterns to understand?

Very difficult  Difficult  Easy  Very easy

As shown in Figure 7-28, only 8 percent of the subject found process patterns not useful at all. Out of 92 percent that found process patterns useful, 24% found them slightly useful, 40% moderately useful, and 28% very useful.

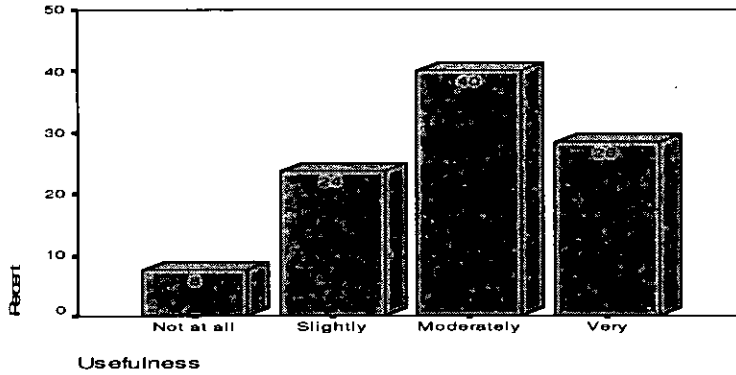


Figure 7-28 process pattern usefulness

Figure 7-29 shows that 4% of the subject found process patterns *very difficult* to understand, and 22% *difficult* to understand. Most subjects said that they found process patterns either easy (61%) or very easy (13%).

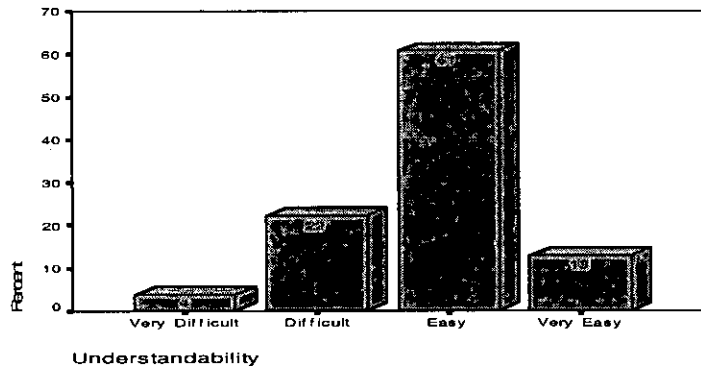


Figure 7-29 Process patterns usability

### 7.9 Summary

The results of the metrics were presented and statistically analysed for statistical significance in this chapter. For each metric, it was determined whether there was a statistically significant mean difference between the treated and control groups caused by the employment of process patterns. It was further analysed whether there were any differences between the group projects and individual projects, for each metrics that indicated a more prominent effect of process patterns on a particular project type (i.e. group or individual). The correlation between each metric and the numbers of logins to the process pattern online resource (treatment) was also analysed to determine if higher usage of process patterns correlated with improved performance in terms of metric results.

For the majority of the metrics, a significant mean difference between the treated and control groups were detected, indicating that the treated groups performed better on those attributes. It was also found that there were correlations between the number of logins and metrics values for the majority of the metrics that showed an effect of process patterns.

---

As well as the results of the metrics and measurements acquired through the measurement process, the marks awarded to the projects, by tutors, were also equally analysed. The results showed that for three ('design and analysis', evaluation, 'project management') out of the four attributes marked, there was no statistically significant difference. The results however showed that the treated groups did significantly better than the control groups for the 'product' attribute, indicating that process patterns improved the software product quality.

The results also showed that, for many metrics, as well as the 'product' attribute (as marked by tutors), the effect of the treatment condition was higher on group projects than on individual projects. This indicates that the process patterns have a more prominent effect on team projects than on individual projects for many attributes. The results also showed that the majority of the subjects who used patterns (i.e. treated groups), found process patterns useful and easy to use. In the next chapter, the results will be analysed and discussed.

## Chapter 8 Analysis

---

### 8.1 Introduction

The results of the experiment for each metric tutor mark were presented and statistically analysed individually in Chapter 7. The aim in this chapter is to analyse the results. There will be a concise representation of the results presented in the previous chapter. There will also be a discussion of the metrics and tutor marks and their corresponding software attributes, which were affected by process patterns. There will also be a discussion of the metrics that showed process patterns had a more significant effect on group projects than individual projects. The software attributes examined, which were not affected by process patterns, will also be listed and discussed. Based on the analysis of the results, there will also be a discussion as to whether the research hypothesis is accepted or rejected. In the final section of the chapter, there will be a discussion of the validity and generalisation of the overall results.

### 8.2 Concise Results Representation

Having discussed each metric and tutor mark in detail in the previous chapter, in this section, Table 8-1, presents a concise representation of the overall results, listing all metrics and tutor marks analysed in the experiment. For each metric and mark Table 8-1 shows whether the following statements are true (✓) or false (✗):

- Process patterns had a significant positive effect on group projects
- Process patterns had a significant positive effect on individual projects
- The effect of process patterns on group projects was significantly higher
- There was a correlation between the number of logins (to the online process patterns) and the value of the metric/mark.

		Positive Effect on Group Projects	Positive Effect on Individual Projects	Group Projects Performed Better Than Ind. Projects	Correlation: Logins and /Metrics/Marks
Metrics	Req. Analysis	Percentage of traceable requirements	√	√	√
		Percentage of reviewed requirements specification	√	√	√
		Percentage of defects fixed	x	x	x
		Percentage of phase time spent in testing	x	x	x
	Design	Percentage of design document reviewed	√	√	√
		Number of methods per class (Methods per Class Ratio)	√	√	√
		Percentage of defects fixed	x	x	x
		Percentage of phase time spent in testing	x	x	x
	Implementation	Comment density	√	√	√
		Percentage of code reviewed	√	√	√
		Productivity (Implementation phase)	√	√	√
		Productivity (complete development project)	√	√	√
		Percentage of defects fixed	x	x	x
		Defect density	√	√	x
		Percentage of phase time spent in testing	√	√	x
	Delivery	Test case density (Test case per Requirement)	√	√	√
		Percentage of defects fixed	√	√	x
		Percentage of phase time spent in testing	√	√	√
Marks	Design and analysis	x	x	x	
	Product	√	√	√	
	Evaluation (tests)	x	x	x	
	Project management	x	x	x	

Table 8-1 A concise representation of metrics/marks results

The results depicted in Table 8-1, show that the majority of the metrics indicate positive effects of process patterns for both group and individual projects. In all the cases that showed a positive effect, both group and individual projects were affected (i.e. there are no cases where there is only positive effect for one project type and not for the other). It is also noticed that in majority of the cases that showed a positive effect, the effect was higher on group projects than individual projects. This indicates that process patterns have a more prominent effect on team projects than individual projects. The results also show that, apart from one exception, there was a correlation for all the metrics that showed an improvement due to the use of patterns, between the number of logins to the online patterns and the value of the metric/marks. This suggests that higher usage of process patterns corresponds to more favourable metric values and, therefore, to better associated attribute quality.

In the following section, the results are further discussed and analysed.

### 8.3 An Analysis of the Results

As depicted in Table 8-1, there are a number metrics that are deemed significant in indicating that, the employment of process patterns had a positive effect on the software attributes, which they measured. Thirteen out of the eighteen metrics investigated showed a statistically significant difference between the treated and control groups. The significant effect of each metric was analysed through sensitivity analysis as described in Chapter 7. The sensitivity margins represent the percentage of change that could take place in the metric's parameters before its conclusion was affected. Table 8-2 lists all the thirteen metrics that showed an improvement as a result of using patterns, in order of the significance of their effect, as determined by the sensitivity margins. The attribute most affected was the 'source code review' with an overall sensitivity margin of 44.7%. The least affected was the 'defect removal ratio (measure by percentage of defects fixed) in the

Delivery phase, with an overall sensitivity margin of 10.2%. There were also a number of metrics which indicated that process pattern usage had no significant effect on the associated attributes. These are listed in Table 8-3.

No.	Metric	Sensitivity Margin %		
		Group Projects	Indiv. Projects	Mean
1	Percentage of source code reviewed	47.5	41.9	44.7
2	Comment density	48.8	37.6	43.2
3	Percentage of phase time spent in testing (Implementation)	41.2	38.4	39.8
4	Defect density	39.2	37.3	38.2
5	Percentage design document reviewed	38.1	32.1	35.1
6	Productivity (Implementation)	33.3	23.8	28.6
7	No. of methods per class (Methods per Class Ratio)	33.8	20.3	27.1
8	Productivity (overall)	27.1	19.2	23.2
9	Percentage of phase time spent in testing (Delivery phase)	29.0	12.1	20.1
10	Test case density (Test case coverage)	19.4	13.0	16.2
11	Percentage of reviewed requirements specification	20.6	10.1	15.7
12	Percentage of traceable requirements	19.1	11.1	15.1
13	Percentage of defects fixed (Delivery phase)	11.5	8.8	10.2

**Table 8-2** Metrics that showed positive effect of process patterns and their effect size

No.	Metric
1	Percentage of phase time spent in testing (RA Phase)
2	Percentage of defects fixed (RA Phase)
3	Percentage of defects fixed (Design Phase)
4	Percentage of phase time spent in testing (Design phase)
5	Percentage of defects fixed (Implementation)

**Table 8-3** Metrics that showed no significant effect of process patterns

As indicated in Table 8-2, patterns have been shown to have a positive effect on the following software attributes:

- Requirements Traceability:** This attribute refers to the ability to describe and follow the life of a requirement, in both a forwards and backwards direction [Ramesh and Jarke 2001]. A requirement should be linked to a higher level document, which could be a higher-level system requirement, as well as downward to the design elements, source code, and test cases that are constructed to implement and verify the requirement [Davis 1993][Hull et al. 2005]. Therefore, higher requirement traceability is desired in a software development project. This attribute was measured through the 'percentage of traceable requirements' metric which showed a significantly higher value for the projects that used process patterns. It has been shown that the employment of process patterns in software development projects improves the requirement traceability by 15.1%. The use of patterns can therefore be said to make a statistically significant improvement to the traceability of requirements.
- Reviews:** Reviews are the most widely used approach for assessing software quality [Sommerville 2007]. Furthermore, inspection of requirements and design are more effective than testing [Hinkle 2007]. An error detected within the development process is 10 to 100 times less costly to fix, than a bug found during the application's operation [Boehm and Basili 2001] [Standish Group 2007]. Boehm [1981] and Jones [1996] have shown that peer review has the most significant defect filtering effectiveness rate of 65% (higher than unit testing of 60%). Therefore, the higher the percentage of the reviewed artefacts, the better the quality of the review process and better the chance of finding any defects [Fagan 1976]. This attribute was measured using the following three metrics: Percentage of reviewed requirements specification, percentage of design document reviewed, and percentage of source code reviewed. The metrics showed a significantly higher value for the projects that used process patterns. It has been shown, through sensitivity analysis, that the employment of process patterns in software development projects effectively improves the reviews process by an average of



31.8% (i.e. Code rev. 44.7%, Design doc rev. 35.2%, and Req. spec. rev. 15.7%). The use of patterns can therefore be said to make a statistically significant improvement to the review process.

- **Granularity/Complexity of modules:** An application developed with more finely granular objects (i.e. lower methods per class, is likely to be more easily maintained and reusable. A larger number of methods per class are likely to hinder extensibility and complicate testing due to the increased object size and complexity. Furthermore, the larger the number of methods, the more complex the inheritance tree and the more limiting the potential reuse and therefore the number of methods per class should be kept as low as possible [Pressman and Ince 2000]. This attribute was measured through the 'Methods per Class Ratio' metric. This metric was first proposed by Chidamber and Kemerer [1994], named Weighted Method per Class (WMC), as a measure of complexity (see Section 4.9). The metric produced results that were more favourable for projects that used process patterns. It has been shown, through sensitivity analysis, that the employment of process patterns in software development projects effectively improves the granularity-complexity modules (classes) by 27.1%. The use of patterns can therefore be said to make a statistically significant improvement to the granularity/complexity of modules.
- **Comment Density:** The comment density metrics is useful for estimating the quality of the code [Lorenz and Kidd 1994]. The higher the percentage of the code that is commented, the better the quality of code in terms of readability, modifiability, and maintainability. It is generally recommended that there should be as many lines of comments as lines of code [Ambler 1998]. The comment density attribute is measured as the ratio of the lines of comments per lines of code (i.e. Locom / LOC). The metric showed a significantly higher value for the projects that used process patterns. It has been shown that the employment of process patterns in software development projects improves the comment density by 43.2%. The use of patterns can therefore be said to make a statistically significant improvement to the comment density.
- **Productivity:** Productivity evaluation is difficult and controversial, and even advice offered by ISO 15393 on productivity measurements have been shown to be misleading [Kitchenham and Colin 2007]. Difficulties in productivity measurement are partly due to the diverse and differing ways and views on how input and output should be measured and the difficulty in measuring them [Kitchenham and Mendes 2004] [Shepperd 1996] [Walton and Felix 1977]. For example, LOC as a measure of output does not take into account many attributes such as verbosity of the programmer, the programming language, and environmental complexity such as skills, pressure, tool support, and computing platform (See Section 4.10). However, LOC and Function Point counts are the most common output measurements used [Maxwell and Forselius 2000]. While some argue that it is unsafe to measure productivity as a ratio of two unrelated variables [Kitchenham and Colin 2007]], productivity as size over effort ratio is by far the most popular method of evaluating productivity. In a literature review of the productivity measurement, Kitchenham and Mendes [2004] found that (with the exception of one) all the surveyed papers to use this method of productivity evaluation. The method of productivity measurement employed in this research is also size over effort, where size is measured in terms of the number of lines of code (LOC), and effort in terms of person-hour. Although imperfect, this method of productivity measurement (i.e. LOC/Effort) is widely used and provides a consistent measure of productivity [MacCormack et al. 2003]. In this study, LOC measurement method and other related factors and issues (e.g. verbosity of the programmer, the programming language, and environmental complexity such as skills, pressure, tool support, computing platform), are randomly spread amongst the control and treated groups, and have therefore neutralised effect. LOC over Effort is therefore deemed appropriate for this study as a way of comparing treated and control groups in terms of their productivity.

Productivity was measured for both the Implementation phase and the complete development project. It was evaluated as size (LOC) over effort (time) where time was measured in terms of person-hour, rather than day, since the subjects often spent part of the day working on the projects. The results can however be easily extrapolated to 'day' as the unit of time (taking a day to be eight working hours) by multiplying the results by 8. For example, productivity of 6 LOC per hour would be equivalent to 48 LOC per day. It has been shown that the employment of process patterns in software development projects improves the productivity in the Implementation phase by 28.6% and the overall productivity (for the complete development project) by 23.2%. The use of patterns can therefore be said to make a statistically significant improvement to productivity.

- **Defect Density:** This metric is generally used in industry for many purposes such as identifying candidate components for further review, or analysing and tracking the impact of defect removal on quality improvement [Ebert 2005]. It is the most commonly used means of measuring quality of a piece of software code and has become the de-facto industry standard measure of software quality [Fenton and Pfleeger 1997]. It is measured as the number of defects detected per LOC ratio. The metric showed a significantly higher value for the projects that used process patterns. A reduction in the defect density in source code is important, especially as studies have shown that up to 65% of defects occur at the design and coding stages [Boehm 1981][Jones 1996]. It has been shown that the employment of process patterns in software development projects improves the defect density by 38.2%. The use of patterns can therefore be said to make a statistically significant improvement to the defect density.
- **Test time allocation (Implementation and Delivery phases):** A right proportion of the phase time allocated to testing is important in providing the necessary time for carrying out the required testing tasks adequately. A small proportion of the phase time allocated to tests would indicate a deficiency and inadequacy in carrying out the test tasks properly. Normally between 30 to 40 percent of project effort is spent on testing [Pressman and Ince 2000]. It is generally recommended in the literature that in most cases, between 30 to 50 percent of the development effort should be allocated to testing [Six sigma][Huang 2004]. This attribute was measured through the 'Percentage phase time spent in testing' metric, which showed a significantly better value for the projects that used process patterns. It has been shown that the employment of process patterns in software development projects improves time allocation to testing by 40% and 20% in the Implementation and the Delivery phases respectively. The use of patterns can therefore be said to make a statistically significant improvement in allocating appropriate test time to the Implementation and Delivery phases.
- **Test Case density (test case coverage):** Every requirement should have one or more tests associated with it [Laplante 2007]. This attribute was measured by the 'test case density' metric, which is evaluated as the ratio of (No. of defined test cases) per (No. of requirements). A higher test case per requirement ratio denotes a more thorough and comprehensive test process, as it offers a higher probability of detecting any defects. The metric results showed a significantly higher value for the projects that used process patterns. It has been shown that the employment of process patterns in software development projects improves the test case density by 16.2%. The use of patterns can therefore be said to make a statistically significant improvement to the test case coverage.
- **Defect removal ratio (Delivery phase):** Defect control and management is crucially important in software development, as defects are a root cause of software failures [Jones 2007]. Therefore, a development process in which more of the detected defects are fixed is more likely to produce a reliable software product. This attribute was measured by the 'percentage of defects fixed' metric, which is evaluated as the ratio of (No. of defects fixed) per (No. of defects detected). This metric is used to provide an indication of the quality of defect correction process and product, by assessing the percentage of the defects that were fixed for each development phase. A higher value would indicate a better defect correction process as well as a less erroneous product. This attribute was measured as the ratio of the number of defects fixed per the number of defects detected. The results of the metric showed a significantly higher value in the Delivery phase for the projects that used process patterns. It has been shown that the employment of process patterns in software development projects improves the defect removal ratio by 10.2% in the Delivery phase. The use of patterns can therefore be said to make a statistically significant improvement to the defect removal ratio in the Delivery phase.

While patterns have been shown to improve the attributes outlined above, the results indicate that they do not make a significant difference to some attributes. These attributes are listed and described below.

- **Test time allocation (Requirement Analysis phase):** The test time allocation attribute was described above. The results showed that there was not a statistically significant mean difference between the treated and control groups in terms of the amount of time they allocated to testing in the Requirement Analysis phase. It can therefore be concluded that process patterns do not affect the proportion of the RA phase time that is allocated to testing.
- **Test time allocation (Design phase):** The results showed that the allocation of test time in the Design phase was not significantly affected by the use of process patterns. It can therefore be concluded that process patterns do not affect the proportion of phase time Design phase time that is allocated to testing.

- **Defects removal ratio (RA phases):** The defect removal ratio attribute was described above. The results showed that there was not a statistically significant mean difference between the treated and control groups in terms of the ratio of the 'number of defects fixed' per 'number of defects detected' in the Requirement Analysis phase. It can therefore be concluded that process patterns do not affect the defects removal ratio in the Requirement Analysis phase.
- **Defects removal ratio (Design phases):** The defect removal ratio attribute (described above) for the Design phase, was not affected by the use of process patterns. It can therefore be concluded that process patterns do not affect the defects removal ratio in the Design phase.
- **Defects removal ratio (Implementation phases):** The defect removal ratio attribute for the Implementation phase was also not affected by the use of process patterns. It can therefore be concluded that process patterns do not affect the defects removal ratio in the Implementation phase.

A trend can be observed in the improved attributes listed above. The trend points to significant improvements to attributes in three general areas of, tests, reviews, and defects as a result of using process patterns. The improvements in testing activities are substantiated by metrics (10), (9), and (3) [Table 8-2], which show significant improvements in test related attributes. The improvements in reviews are substantiated by metrics (1), (5), and (11) [Table 8-2], which show significant improvement in the reviews of the source code, design, and requirements specifications. The improvements in defect control are substantiated by metrics (4) and (13) [Table 8-2], which also show significant improvement in defect density and defect removal ratio.

It is observed from the results that, although the use of patterns improved test time allocation in some phases, they made no statistically significant difference in others. For example, while the test time allocation attribute was significantly improved in both Implementation and Delivery phases, no improvement is noticed in the cases of the Requirement Analysis and Design phases. A closer look at the metric results indicate that both treated and control groups spend a reasonable proportion of the RA phase time (22%) on testing and review. The proportion of phase time spent on test was much lower in the case of the Design phase being around 8% for both the treated and control groups, which is in general much too low. In both cases, the pattern usage does not appear to have made any significant effect. The results point to the conclusion that process patterns do not affect the proportion of phase time allocated to testing in either RA or Design phases. However, it is important that there was a significant improvement (39.8%) in the allocation of time in the Implementation phase, which involved testing and validating the developed source code, which is the backbone of the completed application. It is further significant that there was also an improvement (20.1%) in the allocation of test time in the Delivery phase, which involved 'test in the large' activities that tested the completed software application.

It is also observed from the results that the use of process patterns had no significant effect on the *defect removal ratio* attribute on three of the four development phases (i.e. RA, Design, and Implementation phases). The results however, show that there is an improvement in the Delivery phase for this attribute as a result of using process patterns. Considering that defect removal ratio in the Delivery phase concerns the proportion of the defects corrected in the completed application, improvement (10.2%) in this attribute, as a result of using patterns, is significant in terms the quality (i.e. reduced defects) of the end product.

The trend, which emerges from the results, indicate that process patterns have the least effect on both, the defect removal ratio and the test time allocation attributes, on the first three development phases, while having a significant effect for both attributes on the last phase (i.e. Delivery). The trend further points to improvement in many attributes such as requirement traceability, reviews, comment density, defect density, granularity-complexity, and test case density. An important trend in improvement is noticed on productivity. The results showed that process patterns significantly improved productivity at both phase level (Implementation 28.6%) and project level (23.2%). The higher improvement level on productivity at the Implementation phase indicate that developers were more productive in this phase due to availability of better design documents generated in the Design phase. The developers could concentrate their efforts on the implementation of the prepared design, rather than spending portion of their time on the design implication of the code, and therefore be more productive in the Implementation phase.

The Table 8-1 shows that there have been improvements in all the four evaluated phases of the development lifecycle through improvement to at least two or more attributes in each phase. For example in the Delivery phase, all the three measured attributes showed improvements as a result of using patterns. The Design phase was also improved in terms of smaller granularity of objects, as measured by 'Method per Class ratio' metric.

The Implementation and Requirement Analysis phases were also improved through improvements in attributes such as reviews, requirements traceability, defect density, and comment density.

As detailed in Chapter 7 and listed in Table 8-1, there are a number of metrics, which indicate that process patterns have a more significant effect on group projects than on individual projects. These metrics are listed in Table 8-4.

No.	Metric
1	Percentage of traceable requirements
2	Percentage of reviewed requirements specification
3	No. of methods per class (Methods per Class Ratio)
4	Percentage of design document reviewed
5	Comment density
6	Percentage of code reviewed
7	Percentage of phase time spent in testing (Implementation)
8	Productivity (in Implementation phase)
9	Productivity (overall)
10	Test case density (Test case per Requirement)
11	Percentage of phase time spent in testing (Delivery)

**Table 8-4** Metrics that showed process patterns had a more significant effect on group projects than on individual projects

Based on the metrics results listed in Table 8-4, patterns have been shown to have a more significant effect on group projects than on individual projects on a number of attributes. These are:

- Requirements traceability
- Reviews
- Granularity of modules
- Comment density
- Productivity
- Test time allocation (Implementation and Delivery phases)
- Test case density (Test case coverage)

The improved values in the group projects for the above attributes indicate that process patterns are more effective in team projects, where a number of individuals are directly involved in the project. Studies have shown that effective communication between team members is an essential ingredient of successful software projects [Futrell et al. 2002]. The reason for the improved effect of process patterns on team projects could therefore be due to the influence of process patterns in producing communication within teams that are more effective. This explanation is plausible due to that fact that design patterns have been shown to improve communication within teams [Beck et al. 1996] [Hahsler 2005] [Unger and Tichy 2000]. The survey conducted in this study (see Chapter 3, Table 3-8) also showed that 61% of respondent pattern users believed that pattern improved communication between development team members. It therefore appears that process patterns also have a positive effect on improving communication within teams. There were also a number of metrics (i.e. the defect density and the percentage of defects fixed), which showed that while process patterns had a positive effect, the effect was not more prominent in either project types.

In the case of tutor marks, the four attributes marked were *design and analysis*, *evaluation*, *product*, and *project management*. In analysing the marks with regards to any difference between the treated and control groups, no significant differences were found for three of the four attributes (as shown in Table 8-1). These are 'design and analysis', 'evaluation', and 'project management'. There were however differences between *treated* and *control groups* for the 'product' attribute for both group and individual projects. The results show that the treated groups received significantly higher marks for the 'product' attribute than the control groups, with sensitivity margin of 14.1% as shown in Table 8-5, indicating that the product in the case of the treated groups was of a higher quality. Considering that the product attribute of the project, as marked by the tutors, is a measure of the quality of the delivered software product, it is significant that process patterns have been shown to improve the quality of this attribute. It can therefore be deduced that the application of process patterns improves the quality of the delivered product. Furthermore, it has been shown that the mean difference for the product attribute, between the treated groups and control groups, was higher in the group projects than the individual projects. This indicates that group projects were more affected by the treatment (i.e. process patterns) than the individual projects for the

product attribute. It can therefore, be concluded that the employment of process patterns is more effective on team projects than individual projects in producing a better quality product.

No	Tutor Marks	Sensitivity Margin %		
		Group Projects	Ind. Projects	Mean
1	Product	15.2	12.9	14.1

Table 8-5 Tutor mark attribute, which showed positive effect of process patterns

## 8.4 Research Hypothesis

The research question and hypothesis was discussed in the Section 5.2. The research's *null hypothesis* is:

**H<sub>0</sub>** Application of process patterns in the management of a software development project *will not* improve the quality of the project.

The research's *alternative hypothesis* is:

**H<sub>1</sub>** Application of process patterns in the management of a software development project *will* improve the quality of the project.

In order to test this hypothesis a number of software attributes, across the four major phases of a development lifecycle, were measured and evaluated through an experimental research method. As shown in Table 8-1, there were at least two or more attributes in each development phase that showed statistically significant improvement as a result of employing process patterns in software development projects. It can therefore be said that process patterns improve the overall quality of a software project. Based on these findings, the null hypothesis H<sub>0</sub> is rejected and therefore the alternative H<sub>1</sub> is accepted.

## 8.5 A Discussion of the Results

There are three main possibilities for the reason that majority of metrics showed a significant difference between the treated and control groups in favour of the treated groups. These three are:

1. The treated groups knew that they were part of an experimental study and made extra effort (more than they normally would) to do better and perform to the expectation (the Hawthorn effect).
2. The treated groups knew that they were expected to follow the solutions provided by the process patterns and therefore exaggerated their measurement data to conform to the expectation that patterns improved performance.
3. The treated groups implemented the solutions provided by the process patterns, which resulted in higher quality values for the measured attributes.

The Hawthorn Effect [Parsons 1974] is an important aspect of an experiment design, which should be considered carefully at experiment design level. In the design of the experiment, for this research, the Hawthorn effect was fully considered as discussed in the research method chapter (Chapter 5). Research in software engineering has shown that people can tailor their behaviour to the things they are measured against, and produce the expected results [Weinberg and Schulman 1974]. While the students had to be told that they were participating in an experiment for ethical reasons, they were not told whether they were in the control or treatment groups. Therefore, any Hawthorn effect would have applied to both the control and treated groups and would therefore not affect the outcome of the experiment to compare treated and control groups. Scenario (1) above is not therefore applicable.

The researcher checked the measurement data, wherever possible during the evaluation of projects. Except in very few discrepancies, which could be due to genuine mistakes, the data measurement provided by the subjects corresponded with the project documents. Based on this, it is safe to deduce that measurement data that could not be verified by the researcher, such as time spent in a phase, no. of defects fixed, percentage of source code reviewed, etc. were also reliable and valid. Furthermore, any assumption of the exaggeration of measurement data would also be equally applicable to the control groups, since the control groups were also aware of their participation in the experiment and the importance of the measurement data they provided. Neither groups were

told whether they were in the control or experimental groups. Therefore, even if such exaggeration did take place to some extent, it would not have significantly affected the results of the experiment since such exaggeration would have affected both treated and control groups. Therefore, scenario (2) is also not applicable.

The only significant differential elements between the treated and control groups in the experiment were the application of process patterns (treatment condition). While there were differing abilities and characteristics (i.e. intelligence, hard-work) between the subjects, such extraneous variables were randomly dispersed across both treated and control groups, due to the random nature of the control and treatment groups. Therefore, any significant difference between the treated and control groups can only be due to the introduction and employment of process patterns by the treated groups in their development projects. This factor was further emphasised by the positive correlations found between pattern usage and metric values. Therefore, the difference in metric values between the treated and control groups, can only be attributed to the use of process patterns by the treated groups, as stated in scenario (3).

### 8.5.1 Official Evaluation

Project tutors carried out the official evaluation of the projects in order to mark the projects. However, their evaluation showed a statistically significant difference between the treated and control in only one of the four development attributes marked. That does not correspond with the metric evaluation, in which the majority of metrics showed statistically significant difference in favour of the treated groups. This may be due to two reasons: 1) A difference in the evaluation methods, and 2) A difference in the marking criteria.

While the metric evaluation method focussed on small and specific aspects of the development, such as defects and review quality, the official method of evaluation was more generic, combining a number of attributes. This would have caused a dilution of the effect that could have led to any difference to be less prominent. For example, there was just a single mark for 'analysis and design', which may be too big an attribute to be evaluated by this method. The evaluation scheme (marking scheme) employed by the tutors could have also made a difference. For example, marking criteria and scheme for evaluating 'design and analysis' for grades, 'good' and excellent is as follows: Good = 'Evidence of analysis and design in respect to the original problem', Excellent = 'Analysis and design is explicit. All problems addressed' (See Appendix A. Experiment Details, for marking scheme details). The adopted 5-point scale may have not been sensitive enough to detect the differences between the treated and control groups that would have otherwise been detected by the conducted measurement process.

It is however significant that, for the 'product' attribute, which evaluated the quality of the produced software, there was a significant difference between the treated and control groups in favour of the treated groups. This indicates that the use of process patterns had a positive effect on the quality of the developed software.

### 8.5.2 Generalisations of the Results

The generalisation of the results is directly related to the external validity of the experiment design [Christensen 2006], which was discussed in the research method Chapter 5. The results presented and discussed have shown a clear positive effect for many software attributes measured through specific metrics. The question is whether these results can be generalised to apply to other settings, situations and circumstances. For example, the experiment has shown that the use of process patterns increased *comment density* in the tested samples. Does it follow therefore that the use of process patterns increases 'comment density' in general?

The statistical significance analysis carried out on the sample population of the final year, software development, undergraduates, showed that the mean difference between treated and control groups was statistically significant for the majority of the metrics as listed in Table 8-2. Therefore, such results would apply to the whole population from which the sample was selected (i.e. final year computing undergraduates). Therefore, that makes it statistically safe to state that process patterns (i.e. the treatment) improve the quality of requirement analysis in final year, undergraduate, software development projects.

The results however can be further generalised to apply to the whole software community, if it can be shown that the sample selected is a sampled representation of the software community population. As the software community includes a population of professional software developers/engineers from which no samples were directly taken, it appears that the results may not be statistically applicable to the software community in general. There is however, an argument that the final year undergraduates already have some years of software engineering experience (some at professional levels) and are close to being professionals, and may therefore be

---

considered as a valid sample of the software development community [Carver 2003] [Sjoberg 2002]. Students are widely used as samples/subjects in software engineering experimental research, and it is generally accepted, within the software community, that the results achieved in such experiments can be generalised [Sjoberg et al. 2005]. Therefore, based on the arguments presented in this section, the results and the conclusions achieved in this experimental study are generalised to be universally applicable and acceptable.

## 8.6 Summary

In this chapter, the results presented in the previous chapter were analysed. It has been shown that patterns have a positive effect on the following software attributes:

- Requirements Traceability
- Reviews
- Granularity of modules
- Comment Density
- Defect Density
- Productivity
- Test time allocation at Implementation and Delivery
- Test case density (coverage)
- Defect removal ratio (Delivery phase)

Furthermore, the results showed that process patterns have a more prominent effect on group projects than the individual projects for the following attributes:

- Requirement traceability
- Reviews
- Granularity of modules
- Comment density
- Productivity
- Test time allocation (Implementation and Delivery phases)
- Test case density (test case coverage)

The results of the official tutor marks for the four marked attributes were also analysed. It was shown that the employment of process patterns in software development projects improved the 'product' attribute. This indicates that pattern usage improves the overall quality of the delivered software application.

Based on the results presented and analysed in this chapter, which showed the application of process patterns improved a number of software attributes, the research's alternative hypothesis was accepted. Therefore, the study confirmed that the application of process patterns in the management of a software development project will improve the quality of the project (at the very least in terms of thirteen different quality attributes measured in this experiments and of the overall perception of the quality of the marked product as assessed by the independent markers).

---

## Chapter 9 Conclusion

---

### 9.1 Introduction

The main aim of this research has been to investigate the utility and value of process patterns in the management of software development projects. This chapter offers a summary of the main concepts, outlines the research contributions, and provides a discussion of the results. The chapter contains the following sections:

- Summary of main concepts
- Research contribution
- Results summary
- Limitations
- Research's Impact
- Future Work

In the next section, the main concepts and topics concerned in this research will be briefly discussed.

### 9.2 Summary of Main Concepts

The pattern concept, originating in architecture, has penetrated many areas of software engineering. Patterns are currently employed in many domain and technology areas such as distributed computing, security, object-domain-aspect oriented development, embedded systems, and development process [Buschmann et al. 2007]. While the pattern concept has been adopted and applied in many domains of software engineering, there are many issues about patterns that are being discussed within the pattern community. Some criticise the patterns for being vague in terms of their structure and context, and call for the pattern concept to be formalised to be more useful [Bayley and Zhu 2007][Taibi and Ngo 2001]. However, while formalisation would make it easier to create pattern tools to assist with indexing, searching, and mining patterns, it would make the pattern concept too restrictive, causing it to lose the flexibility and the abstract nature that is its fundamental characteristic. Furthermore, although formalisation of patterns may be possible for some well-defined domains, such as software design and architecture, it would be extremely difficult to implement fully in some areas of software engineering such as development processes. Process patterns typically involve a human element, making them unsuitable to be strictly formalised in any comprehensive manner. Fundamentally, both elements of process pattern (i.e. development processes and patterns) are abstract to an extent, and are therefore inappropriate to attempt to fully formalise and automate their extraction or implementation (see sections 2.6.4 and 2.3.4).

A further important issue with software patterns is, that while there are numerous software patterns stored in a number of software pattern repositories [Booch 2008][Portland Pattern repository], it is difficult for pattern users and software practitioners to find appropriate patterns that are best suited to the problem they are trying to solve [Kampffmeyer and Zschaler 2007]. This is partly due to inappropriate search mechanisms. Furthermore, the pattern repositories do not document how a number of patterns can be used in a sequence, to solve more intricate and complex problems. The true power of the pattern concept is in the way a number of patterns can interlink, and collaborate in a sequence to solve a problem. While some research is being done in this area (for example [Siddle 2007]), this important aspect of patterns has not yet been researched and utilised in any significant way. Software patterns can become much more useful and more widely used once considerable progress is made in this area.

The value of the patterns concept as a method of capturing 'best practice' in the software development processes was realised and documented early in the introduction of patterns in software engineering, in works such as [Coplien 1995][Whitenack 1994][First PLoP conference Proc. 1994]. However, the pattern community and software practitioners have by large concentrated on the product aspects and design utilities of the pattern concept [e.g. design patterns]. There is currently far fewer published work on process-based patterns than on software design and architecture-based patterns. The usage of development process-based patterns in the software development industry has been much lower in comparison to design-based patterns. One of the reasons for the low usage of process patterns in industry could be because formal development methodologies and processes are little understood and practiced in many immature software development companies. Some studies



have shown 35% of software development organisations have an ad hoc, individual-based, and informal development process in place [Yourdon 2008]. There has also been a dearth of empirical research assessing the utility of patterns in software engineering. While there have been a few empirical studies to evaluate the effect and value of design patterns on some aspects of software development [Unger and Tichy 2000] [Prechelt 2001, 2002], there appears to be no credible empirical studies to investigate the utility and value of process patterns. In addressing this issue, it was the main objective of this research to investigate empirically whether the application of process patterns, in a software development project, improved its quality. As a result, the research provided evidence that the application of process patterns has a positive effect on many attributes, such as, traceability of requirements, defect density, productivity, comment density, etc. These will be briefly discussed in the results section 9.4). This research has shown that process-based patterns can play an important part in improving software development practice, by enhancing the quality of many software development attributes.

Measurement is essential in producing tangible evidence in any field of science and engineering. While in some fields, such as physics, measurement is a mature and well-understood process of crucial importance, it is much less understood in software engineering and therefore its role has not been anywhere as significant [Ebert and Dumke 2007]. There are those within the software community who argue that proper and accurate measurement in software engineering is currently impossible, because software engineering itself is not yet fully understood. It appears that software maturity and measurement are closely interlinked in that accurate measurement will only be possible once software development practice is fully matured. Conversely, software development will not reach full maturity until it can be accurately measured. However immature and flawed, measurement endeavour has to be continued earnestly in software engineering, if not for the usefulness of the results that they currently produce, at least for their value in the advancement of our understanding of software itself, and the ways that it can be measured. The measurement concept and practice has been an important and integral part of the controlled experimental research method devised for this research. Without software measurement concepts and related metrics, it would be impossible to evaluate software attributes and therefore impossible to conduct such experimentation to assess the utility of a concept, entity, or technology. Research works, such as this project, play an important part in edging forward our understanding of software and its measurement concepts. One feature of this research is that it has contributed to our understanding of the empirical application of software measurement, by employing it in an experimental research. The research provides a demonstration of the way the measurement concepts and theories can be employed in experiments to evaluate software attributes. Such research works, not only produce valuable measurement for evaluation of software artefacts and attributes, but also are useful in highlighting any deficiencies in the software measurement theories and practices.

While in recent years there have been many more empirical studies in software engineering than in the past, the numbers and the quality of the studies in terms of validation still fall short of what is necessary and required, in assisting the advancement of our understanding of software engineering [Sjoberg et al. 2005] [Koziolek 2005]. One of the main reasons for such lack of empirical research in software engineering is the difficulty in performing validated empirical research in the field. While empirical research in basic sciences such as physics and chemistry can be done in a laboratory environment, where variables are perfectly controlled, empirical research in software engineering is much more difficult. Software engineering takes place in the real world and is heavily subjected to human factors and therefore designing and performing validated controlled experiments under these circumstances is at best difficult and at times impossible. Difficulties in providing proper controls of the variables in software engineering experiments could invalidate any results. Consequently, many researchers appear to have shied away from undertaking such research, especially as many journals expect validated and controlled experiments.

Although experimentation in real-life situations involving human subjects is tedious to design and conduct, mainly due to difficulties in providing full control over extraneous variables, experimenters however often encounter reviewers that expect perfection and absolute certainty [Tichy 1998]. Furthermore, there are often difficulties for experimenters to publish their work in journals because many established journals find it difficult to find editors and reviewers capable of reviewing experimental works [ibid]. It is however recognised that the difficulty in carrying out perfectly controlled experiments, or producing perfectly validated results, should not prevent researchers from performing empirical investigations in software engineering. Experiments are conducted in the real world and are always flawed in some way [ibid]. Therefore, researchers should take the opportunity to perform experimental research even though the circumstances may not adhere to the perfect theoretical format. There are many challenges, such as improvement in synthesis of empirical evidence, which requires the cooperation of academia and industry to provide the necessary resources to conduct a greater number of quality empirical research projects [Sjøberg and Dybå 2007]. The empirical investigation, carried out in this research programme, is an attempt to demonstrate a way of performing software experimentation within an academic institution involving live courses. By no means is it claimed that the experiment has been perfectly

controlled and flawless, or that it has produced perfectly validated and generalised results. Realistically the world of software engineering does not currently make that possible. We have nonetheless tried to devise and perform the experiments in this research as objectively and scientifically as possible, given the environmental context and constraints. In addition to its utility in evaluating the effect of process patterns, an important feature of this experiment design is that it provides a method or a technique, for validating software patterns, which can generally be used to validate any software pattern.

### 9.3 Research Contributions

This research programme has made a number of contributions to the scientific body of knowledge, which are individually stated and are briefly discussed in this section.

#### 9.3.1 Key Contribution

*Provision of evidence that the application of process patterns in software development projects improves software project quality*

This is the direct contribution to the research question. Evaluation of process patterns in terms of their effect on software development projects has been the main aim of this study. An extensive two-part controlled experiment, involving over 260 projects and over 750 subjects, spanning two semesters (one academic year) was designed and conducted for this study. Such extensive, comprehensive and high-scoped experiments have been rare within software engineering research. A number of software attributes, measured through metrics, showed that the application of process patterns improved their quality (presented in the results section 9.4). In general, the study showed that the application of process patterns in software development projects, lead to improved quality [Estabraghy and Dalcher 2007a] in at least thirteen of the measured attributes.

#### 9.3.2 Additional Contributions

*Determination of whether the application of process patterns varies in terms of its influence on team and individual projects*

One important and interesting result, that the study produced, was to show that there were differences in the effect of process patterns between group projects and individual projects. The study has provided evidence that process patterns are more effective in team projects, than in single-person (individual) projects. This is further discussed in the results section 9.4.

*Provision of an experimental technique for validating patterns*

The pattern community has introduced the three-rule, which states that a pattern is valid if it is observed in three separate situations. That, however, does not differentiate between 'good' and 'bad' patterns. In other words, a pattern may occur in three different situations but nevertheless be a 'bad' (e.g. inappropriate or harmful) pattern. This study has provided a validation mechanism through experimentation, which could be employed to test the validity ('goodness') of one or more patterns. The experimental design and conduct in this study can be used as a model for others to evaluate the validity and usefulness of software patterns.

*Design and implementation of an experimental methodology in real-life situations*

Experiments in 'real-life' settings are more difficult to design and conduct than those carried out in laboratory settings (controlled environment) where one can achieve relatively full control over variables. Results produced in laboratory settings, however, may not extrapolate to real-life situations in which real people work on real projects. The controlled experiment for this study was carried out in a 'real-life' situation where the experiment subjects (final-year undergraduate students) worked on their computing projects, which acted as the objects of the experiments.

*Provision of an example of carrying out software experimentation in educational establishments using live courses*

Educational establishments and students are valuable resources that can be employed for carrying out empirical research in software engineering. This study has demonstrated that live courses and modules can be fruitfully

employed to carry out experimental investigation in software engineering. The experiment design and process produced in this study can be used as a model or roadmap to carry out further empirical research in such environments.

*Provision of an example of devising a measurement process to evaluate software projects*

While measurement in software engineering is technically challenging and immature compared to other engineering disciplines, it should be an essential component of software development projects in software development organisations. In fact, one important measure of an organisation's competency is the quality of its implemented measurement process. This study has made a contribution in providing an example of devising a tailor-made measurement process for software project evaluation.

*An investigation of issues in architectural patterns and their implication in software patterns*

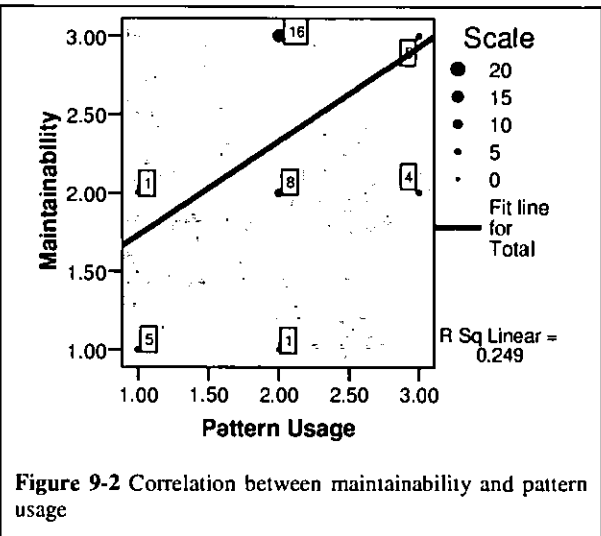
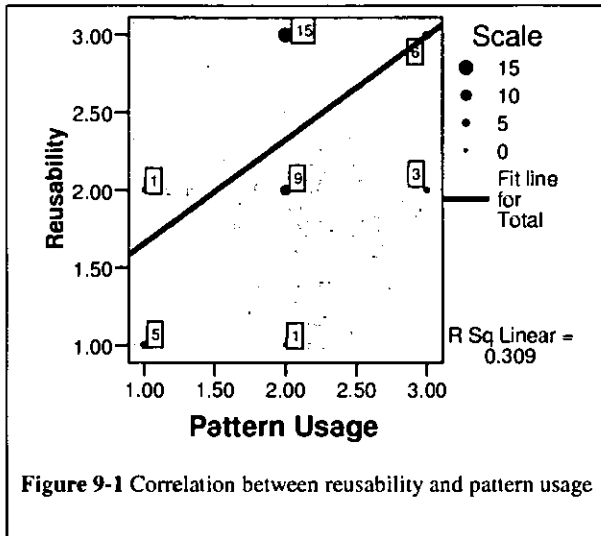
While some architects take the view that the patterns preserve profound designs in architecture and guide architects, most argue that they stifle creativity and invention. Given the concerns and objections architects express about architectural patterns, should the software community be concerned that the pattern issues raised by the architects could at some point catch up with software patterns and render them effectively harmful? In this study, architects within the UK universities were queried for their views on the impact and value of architectural patterns. Based on the results, whether the issues raised in architectural patterns could also apply to software patterns now or in the future, was discussed.

*Provision of evidence of pattern usage rates in the industry*

While there have been numerous books, articles and papers on software patterns in the last decade, few published works produced evidence of software pattern usage-rates in the software development industry. A survey of 67 software development companies in the UK showed that the majority of the surveyed companies (60%) used software patterns. The survey also indicated that by far the most popular software patterns were design patterns, with only 7% of the surveyed companies using process patterns [Estabraghy and Dalcher 2007b].

## 9.4 Summary of Results

Two preliminary surveys were carried out at early stages of this research in order to provide an understanding of how the pattern concept was used in practice, in both its original (i.e. architecture) and software engineering fields, which helped in devising the research question. The architectural pattern survey indicated that architectural patterns suffer from criticisms of being anti-creativity and prescriptive, and therefore enjoyed minimal usage and support within the architectural community. Only 15.4% of the architects surveyed viewed architectural patterns as having a positive effect on architecture. There was shown to be a positive correlation between pattern usage levels and architects' viewpoints. The software pattern survey, however, indicated that software patterns were widely used in software development organisations. The results showed that 60% of the surveyed software development organisations used patterns. The vast majority of these companies, however, used software design-based patterns, such as design patterns, and only 5.7% of them used development process based patterns such as process patterns. The majority of companies that used patterns stated that patterns improved the quality of software attributes such as reliability, maintainability, testability, etc. There were found to be positive correlations between pattern usage and some of these attributes as depicted in scatter plots in Figure 9-1 and Figure 9-2 (see Chapter 3). The survey also showed that only a small proportion of the surveyed companies (6%) developed and published patterns. Of the companies that did not use patterns, 81% said that they were unnecessary, and 88% said that they did not have skilled staff to use patterns. The fact that 81% of the companies that did not use patterns, found patterns to be unnecessary in their practice, indicates that the pattern community has much to do in introducing and publicising patterns. The pattern community should therefore provide more support to encourage software practitioners, to actively learn and improve their pattern skills and, to implement them in their software development practice. Furthermore, the community should demonstrate to the software industry, the benefits that can be gained by using patterns. This research has played an important part in providing evidence of the usefulness of patterns in software development process, which could encourage more software practitioners to employ patterns in their practice.



An experimental research method was devised and implemented to investigate the research question and test the research hypothesis (see Chapter 5). The experiment subjects were divided into two groups of treated and control, where the treated groups used a set of process patterns (covering a complete development lifecycle) to use in their software development projects. The objective was to determine if the use of process patterns as a whole (i.e. not any particular process patterns), improved the quality of their software projects. The experiment, involving two types of software development projects (group and individual), was designed and conducted, through which a number of software attributes were measured using appropriate metrics. The marks awarded by the tutors to four attributes of the projects under investigation were also used in the experiment. The results of the metrics and tutor marks are summarised and presented in Table 9-1.

		Positive Effect on Group Projects	Positive Effect on Individual Projects	Group Projects Performed Better Than Ind. Projects	Correlation: Logins and Metrics/Marks
Metrics	Req. Analysis	Percentage of traceable requirements	√	√	√
		Percentage of reviewed requirements specification	√	√	√
		Percentage of defects fixed	x	x	x
		Percentage of phase time spent in testing	x	x	x
	Design	Percentage of design document reviewed	√	√	√
		Number of methods per class (Methods per Class Ratio)	√	√	√
		Percentage of defects fixed	x	x	x
		Percentage of phase time spent in testing	x	x	x
	Implementation	Comment density	√	√	√
		Percentage of code reviewed	√	√	√
		Productivity (Implementation phase)	√	√	√
		Productivity (complete development project)	√	√	√
		Percentage of defects fixed	x	x	x
		Defect density	√	√	x
	Delivery	Percentage of phase spent in testing	√	√	√
		Test case density (Test case per Requirement)	√	√	√
Percentage of defects fixed		√	√	x	
Marks	Percentage of phase time Spent in Testing	√	√	√	
	Design and analysis	x	x	x	
	Product	√	√	√	
	Evaluation (tests)	x	x	x	
	Project management	x	x	x	

Table 9-1 Summary of the results

Based on the results of metrics and tutor marks depicted in Table 9-1, patterns have been shown to improve a number of software attributes. The improved attributes, and the percentage of statistically significant improvements, have been shown in Table 9-2.

No	Metric	Sensitivity Margin %		
		Group Projects	Ind. Projects	Mean
1	Percentage of source code reviewed	47.5	41.9	44.7
2	Comment density	48.8	37.6	43.2
3	Percentage of phase time spent in testing (Implementation)	41.2	38.4	39.8
4	Defect density	39.2	37.3	38.2
5	Percentage design document reviewed	38.1	32.1	35.1
6	Productivity (Implementation)	33.3	23.8	28.6
7	No. of methods per class (Methods per Class Ratio)	33.8	20.3	27.1
8	Productivity (overall)	27.1	19.2	23.2
9	Percentage of phase time spent in testing (Delivery phase)	29.0	12.1	20.1
10	Test case density (Test case per Requirement)	19.4	13.0	16.2
11	Percentage of reviewed requirements specification	20.6	10.1	15.7
12	Percentage of traceable requirements	19.1	11.1	15.1
13	Percentage of defects fixed (Delivery phase)	11.5	8.8	10.2

Table 9-2 Improved attributes and the effect size

Each attribute listed above is an indicator of the quality of an aspect of a software development project and therefore, improvements in these attributes are sought and desired by software practitioners (see Section 8.3). It is significant that the usage of process patterns in software development projects, improves the quality of many attributes as listed above. These results show that the application of process patterns improve software attributes in development activities such as testing, reviews, and productivity. Furthermore, the Table 9-1 shows that the application of process patterns resulted in significant improvements in software attributes related to the RA, Design, Implementation, and Delivery phases. Significant improvement to many attributes across all the four major development phases of a development lifecycle, has been shown to be an important advantage of using process patterns in software development projects.

The difference between the effect of process patterns on group projects and individual projects were measured. The result showed that for many attributes the treated subjects in group projects performed significantly better than the treated subjects in the individual projects. This indicates that process patterns have a more prominent effect on the group projects than individual projects. It has been shown [Table 9-1] that patterns have a more significant effect on group projects than on individual projects for the following attributes:

- Requirements traceability
- Reviews
- Granularity of modules
- Comment density
- Productivity
- Test time allocation (Implementation and Delivery phases)
- Test case density (Test case per Requirement)
- Product quality

The improved values in the group projects for the above attributes indicate that process patterns are more effective in team projects, where a number of individuals are directly involved. One possible reason for the improved effect of process patterns on team projects could be due to the influence of process patterns on producing more effective communication within teams. It has already been shown that design patterns improve communication between development team members [Beck et al. 1996] [Prechelt 2002] [Unger and Tichy 2000] (see also Table 3-8). Based on the results achieved in this study, it appears that process patterns also have a positive effect on improving communication within teams. Considering the importance of effective communication in projects [Futrell et al. 2002], by improving communication between team members, application of process patterns will play a role in helping to improve the software projects and therefore enhance their chances of a successful completion.

The projects investigated in this study were officially marked by tutors. The four marked attributes of interest to this study were 'Design and Analysis', 'Evaluation', 'Product', and 'Project Management'. The marks were subsequently analysed statistically for any difference between the treated and control groups. There were no significant differences found between the treated and control groups for three of the four marked attributes. The three marked attributes that showed no significant difference were, 'Design and Analysis', 'Evaluation', and 'Project Management'. There were however differences between treated and control groups for the 'product' attribute for both group and individual projects. This showed that the treated groups received a higher mark for the 'product' attribute, than the control groups indicating that the product in the case of the treated groups was of a higher quality. It is significant that process patterns have been shown to improve the quality of the product attribute, considering that it is a measure of the quality of the delivered software product as evaluated by the tutors. Based on the results of this attribute, it can therefore be deduced that the application of process patterns improves the quality of the product. Furthermore, it has been shown that the mean difference between the treated groups and control in group projects, in terms of the product attribute, was higher than the individual projects. This indicates that group projects were more affected by the treatment (i.e. process patterns) than the individual projects for this attribute. Therefore, it can be concluded that the employment of process patterns, is more effective on team projects than individual projects, in producing better product.

The experimental research method was to test the following null hypothesis:

- $H_0$       Application of process patterns in the management of a software development project *will not* improve the quality of the project.

The summary of the results presented in Table 9-1, show that application of process patterns improved a number of software attributes in each of the four major phases of the development lifecycle. Based on these findings the stated null hypothesis is rejected. Therefore, the alternative hypothesis that, the application of process patterns in the management of a software development project will improve the quality of the project, is accepted.

There is always a degree of limitations on any comprehensive research project. This research programme is no exception. The limitations of this research programme are discussed next.

## 9.5 Limitations

In studying real life situations and designing experiments within that environment to learn and understand some phenomenon or test some hypothesis, one has to understand the constraints and limitations involved, and design and conduct the experiment accordingly. Therefore, the nature of this study, as real life experimentation, is such that it invariably brings constraints on the experiment design. The main issues are discussed below:

- Selection of subjects of same abilities to both control and treatment groups
- Control of the amount of treatment condition given to the treatment groups
- Time given to accomplish project tasks
- Variation in team sizes
- Ethical issues
- Experiment scope

The experiment design considered the above points and found solutions in a way that the internal/external validity of the experiment was not adversely affected. These are explained in the following paragraphs.

Students working on group projects had different abilities and characteristics. The students chose their own team members to form a group to work on a common project. Students therefore could not be assigned to teams according to their abilities. It is also generally difficult to judge accurately student abilities according to some criteria and match them. However, due to the large number of groups and their random nature, any differences and discrepancies in the groups were randomly dispersed between the control and experimental groups, and were therefore constant and did not adversely affect the results of the experiment.

It was not possible to control and measure accurately the exact amount of treatment condition (i.e. process patterns) that the subjects used. They were told to use as many process patterns as they needed for their project. However, while the number of times each subject accessed the process pattern resource on the website were recorded as a measure of the usage rate (discussed in Section 5.7.1), the system did not record which patterns were accessed and used by the subject. It would have been advantageous in terms of knowing the access rates of the individual used patterns, had the system recorded such data.

Although there was a set amount of time that the subjects had to work on their project, the amount of time they actually spent on the development activities was based on their own statements. They declared how much time they spent on development activities on an online measurement form. Their estimation of the time spent in each phase was accepted to be approximate to the actual time spent. The subjects were required to fill-in the online measurement forms as accurately and as honestly as they could. They were told that the values they entered on the measurement forms would not have any effect on their marks for their projects. However, because of the random nature of the treated and control groups, any inaccuracies would be equally portioned to both treated and control groups and therefore would not affect the validity of the results.

Although there was normally a team size of five for group projects, the size of groups could change and in some cases, it did. However this did not affect the experiment as time and effort was based on hours (person-hour) spent on the project. Furthermore, any change in the team size affected both treated and control groups, and was therefore a constant factor. Therefore, while a consistent group size would have been preferred, the change did not affect the experiment's objectives or results.

There is always an element of ethical concern in experimentations involving human subjects, which have to be fully considered in the design and conduct of an experiment. Ethical issues concerned with the experiment had to be dealt with head on and from the first principles, meeting requirements such as, fairness, confidentiality, and others as discussed in Chapter 5. The design of the experiment had to be therefore devised in such a way to satisfy the University's Ethics Committee, that all ethical issues were fully considered to prevent a breach of ethics. Compromises had to be made and the experiment design went through a number of revisions before the

committee approved it. The full ethical consideration of the experiment, however, did not cause any serious weaknesses to the experiment design.

The experiment's aim was set to cover a complete development lifecycle, which required an extensive set of process patterns to be used. On reflection, it would have been better if the experiment concentrated on the evaluation of a single phase of the development lifecycle (e.g. Requirement Analysis) and a few related attributes. This would have required the use of a small number (less than 10) of process patterns, related to the investigated development phase, which would have been easier to manage and control. Although the scope would have been limited, the results would have provided the opportunity to scrutinise the effect of the individual patterns. In addition, the devised measurement process, covering a complete development lifecycle, proved too ambitious and excessive in terms of its attempt to collect and analyse a large set of measurement data (62 taken by subjects, and 12 by the researcher, for each project investigated), given the scope of the research. The study of a single development phase would have meant a smaller, but perhaps more detailed set of measurement data, could be collected and analysed.

In the case of the two preliminary surveys, a higher number of participants would have helped decrease the marginal errors. However, in both surveys sufficient numbers of participants took part to reach some generalised conclusions.

## 9.6 Research's Impact

Prior to this research, the utility and value of process patterns in software development was unclear. While there have been many theoretical works on process and organisational patterns, there did not appear to be any empirical studies to evaluate the utility and value of process patterns in improving the quality of software development projects. While patterns have so far had a considerable impact on the design and architecture aspect of software development in the form of software design patterns, their impact on the actual process of developing software has been minimal.

The evidence gathered through a preliminary survey conducted in this research, as well as other studies (for example [Manolescu et al. 2007]), indicate that while software design-based patterns such as design patterns are popular and widely used, development process based patterns, such as process patterns are far less popular and used. It appears that the software development industry requires evidence of the usefulness of process-based patterns before they are will use and integrate them into their development processes. This research has done that by investigating the utility of the process patterns in software development practice, and provided evidence of its usefulness in improving a number of software attributes. Indeed the experiment has shown that patterns do improve the quality of software projects.

As a result of this study, there is now scientific research and relevant data available to the scientific community in general, and the software engineering and pattern communities in particular, on the impact and effect of process patterns in software development practice. The research provides evidence that the employment of process pattern improves the quality of software development projects. More crucially, the research has indicated thirteen specific measures that are improved as a result of applying process patterns. The results show that the application of process patterns improves software attributes across the four major phases of the development lifecycle (i.e. Requirement analysis, Design, Implementation, and Delivery). This research complements other empirical research [Prechelt et al. 2001, 2002] [Unger and Tichy 2000] on design patterns, in evaluating the software patterns and providing evidence of their utility and value. This should encourage software practitioners and the software industry in general to take more notice of the value of process-based software patterns, to implement and employ them in their software development projects.

It is interesting that the survey research conducted in this study provides evidence that the pattern concept has not received much support in the architecture community where it originated. This raises the question, whether the software community would continue to embrace the pattern concept in the future. It is hard to say in the long term, but favourable pattern evaluation results such as this and others, as well as strong qualities, such as its flexibility to adapt to fast changing circumstances in software engineering, should provide software patterns a relatively long lifespan. For the moment, certainly, software design patterns are continuing to rise in popularity within the software community [Buschmann et al. 2007b] and will therefore be with us for a little while yet. The results and conclusions of this research should encourage a wider use of patterns in general and process patterns in particular, making a difference in the way future software development is managed and produced.



## 9.7 Future Work

This thesis has explored many issues in the area of software patterns, software experimentation, and software measurement. Both software measurement and software experimentation concepts were employed in this research project to advance our understanding of software patterns and evaluate their utility and value in software engineering. In the following sections, the areas for future work are discussed in relation to these topics.

### 9.7.1 Software Experimentation

Educational establishments provide immense resources for experimental research. Resources such as courses, modules, and students are invaluable in enabling researchers to pursue research in the field of software engineering. One crucial aspect of tapping into such resources is the understanding of the ethical issues concerned in using students and live courses, as the subjects and objects of the experimental research works. This research demonstrated that, despite ethical and other constraints, such research in software engineering is both possible and feasible. Further research is necessary in understanding the constraints involved, in design and conduct of such experiments, and will provide guidelines and models of how to best design and conduct software experimentation in such environments. There may always be a trade-off between the needs of the experimenter, to find the best solutions to questions or hypotheses, and those of the ethical issues concerned when students are subjects of the experiment. It is down to the experimenter to produce the best experiment design to find the best solutions, while adhering to the concerned ethical issues and regulations. There is much scope for future work in this area.

Future research could undertake to replicate this experiment in an industrial environment. Replicating the experiment in industry involves different types of subjects (i.e. professionals), as well as different settings (work/business environment). Because subjects in such an environment would be professionals rather than students, ethical issues concerned may also be different. It would be useful to determine if similar results or conclusions would be achieved, if the experiment were replicated in an industrial setting.

Measurement in software engineering is still relatively young and immature. While there has been both theoretical and practical work in this field in the last few decades, software engineering measurement is not as much understood as measurements in some other engineering and science disciplines. Software engineering has been changing and progressing rapidly through the advents of new technologies and, therefore, requires new and appropriate methods of measuring and evaluating software to be constantly devised. There remains much more research to be done, in order to gain a better understanding of software engineering, to enable us to devise appropriate measurement theories, principles, and practices, to measure the different aspects of software projects and applications accurately and consistently.

### 9.7.2 Patterns

Experienced practitioners use solutions that have been proven in their experience to work. These are potential patterns that should be extracted and stored in a specifically designed database repository, so that they can be utilised by others. Rising [1998] proposed a number of ways of mining such experience and knowledge in patterns. These techniques include interviewing, workshops, meetings, and classes. There are already many ongoing projects, which are being conducted by the pattern community, that aim to capture and store patterns. One such significant example is Booch's Handbook of software architecture [Booch 2008] containing over 2000 patterns. However, a problem with such pattern repositories is currently the lack of appropriate indexing and search facilities, to enable pattern users to find the specific patterns that would apply and solve the particular problems they are looking to resolve. Furthermore, the current repositories fail to provide and advise on how a number of patterns within the repositories, can be linked in a sequence to solve more complex problems. Therefore, further research needs to be conducted on devising an appropriate and specific system of patterns repositories that would enable such indexing and search facilities.

There are currently many methodologies (e.g. Agile, OPEN, SSDM ...) that are being practiced in industry. Practitioners of such methodologies will learn through repetition and experience the solutions that work and those that do not. Whatever methodology is officially practiced and established in an organisation, it is not fenced with a strict set of constraining rules. The methodology would be flexible and open to application according to the problem at hand. In applying and conforming to the established methodology, practitioners find solutions to process-related problems through time and experience. In mature organisations where development methodology and process is established, expert practitioners have patterns of process solutions, (either in their mind or written) that work. Often such knowledge lives in the 'experts' heads only and are not formally written

---

down and recorded. The pattern concept provides a medium in which such knowledge is recorded precisely and usefully for reuse. There is therefore, an area of research to establish how expert practitioners use their knowledge and experience in providing solutions to development process related problems. Such research can address questions such as what problem solving methods can be used and how the solutions can be captured in the form of process patterns for reuse.

The concept of software patterns needs to be given a fresh look, to concentrate on the human, harmony, and aesthetic aspects, rather than simply apply them as a technical means of capturing and recording software design and experience. The pattern concept has much more meaning and potential and currently only some simplistic aspects of it is being utilised. More research needs to be carried out in this area to establish the harmony and aesthetic focus and aspect of patterns, (as documented by Alexander [1977, 1979] in the field of architecture) in software engineering.

While the terms harmony and aesthetics might seem foreign and out of place in the field of software engineering and are more related to social sciences, the facts suggest otherwise. For example, software engineering involves components that must work in harmony, to establish a perfect system and control mechanisms. Furthermore, as software interacts with humans in one way or another, the aesthetics aspect of it becomes important in providing an environment in which people enjoy using and interacting with software. While there has been much work done on software patterns from a technical point of view, few seem to have focussed on the social aspects of software patterns. Further work needs to be done in this area to understand the link between the pattern concept and the social aspect of software systems.

The survey carried out in this study showed that most architects surveyed, believed that architectural patterns stifled creativity. Further work needs to be done in this area to determine if patterns do hinder creativity in software engineering.

The results of this study indicated that process patterns have a better effect on team projects than on individual projects. Further work needs to be done to determine if the improvement rate is proportional to the size of the teams. That is, whether, as the size of teams using process patterns increases, will there be a proportional effect in terms of its effectiveness?

The pattern concept therefore provides a large area of research in software engineering. While a fair amount of research in this field is already taking place, which has resulted in hundreds of published papers, much more research work remains to be done, focussing especially on other non-technical and aesthetic aspects of the pattern concept in relation to harmonious software systems.

## Reference and Bibliography

---

- Abreu, FB. (1995). The MOOD Metrics Set, Proc. ECOOP'95 Workshop on Metrics
- Adolph, S., Bramble, P., Cockburn, A. (2002). Patterns for Effective Use Cases. Addison Wesley
- Alexander, C. (1999). The origins of pattern theory: The future of the theory, and the generation of a living world. IEEE Software, 16 (5): 71-82
- Alexander, C. (1979). The timeless way of building, New York. Oxford University Press
- Alexander, C. (1977). A pattern language, Oxford University Press
- Alexander, C. (1970). Notes on the synthesis of form, Oxford University Press
- Alexander, C. (1988). The Oregon Experiment, Oxford University Press
- Alexander, C. (1995). The Mary Rose Museum, Oxford University Press
- Alexander, C., Eisenman, P. (1983). Contrasting concepts of harmony in architecture, Lotus International 40
- Ambler, SW. (1998). Software process patterns, Cambridge University Press
- Ambler, SW. (1999). More software process patterns, Cambridge University Press
- Ambler, SW. (2002). Agile modelling, John Wiley and Sons Inc
- Ambler, SW. (2005). The enterprise unified process, Prentice Hall
- American Psychological Association (Code of Ethics). (2002). <http://www.api.org/ethics>
- Annett, J., Duncan, K. (1967). Task analysis and training design. Occupational Psychology no. 41
- Annett, J., Duncan, K., Stammers, R., Gray, M. (1971). Task analysis. London: HMSO
- Annett, J. (2004). Hierarchical task analysis, in handbook of cognitive task design, Diaper (2004) Chapter 3, Mahwah NJ: Lawrence Erlbaum
- Appleton, B. (2000). Patterns and Software: essential concepts and terminology, [www.enteract.com/~bradapp](http://www.enteract.com/~bradapp)
- Appleton, B. (1997). Patterns for conducting process improvement, proceedings of the 4<sup>th</sup> PLoP Conf.
- Aronson, E., Carlsmith, JM. (1968). Experimentation in social psychology, the handbook of social psychology
- Armour, P. (2002). Ten unmyths of project estimation, Comm. ACM, 45(11), 15-18
- Asteen, LA. (1988). The Science of Patterns, Science, 240: 611-616
- Babbie, E. (2001). The practice of social research (9<sup>th</sup> Ed.), Belmont, CA: Wadsworth/Thomson Learning
- Babbie, E. (1990). Survey research methods (2<sup>nd</sup> Ed). Belmont, CA: Wadsworth Publishing Company
- Baker, AL., Bieman, JM., Fenton, NE., Gustafson, D., Melton, A. (1990). A philosophy for software measurement, J Systems Software, Vol 12 , 277-281, July, 1990
- Bansiya, J., Davis. C. (2002). A hierarchical model for object-oriented design quality assessment. IEEE Transactions on Software Engineering, 28(1) pp 4-17

## Reference and Bibliography

- Barnett, V., Lewis, T. (1994). *Outliers in Statistical Data* (3<sup>rd</sup> Ed). Wiley & Sons, New York
- Basili, VR., (2005), *Using Measurement to Build Core Competencies in Software*, Data and Analysis Centre for Software Seminar
- Basili, VR. (2007): "The Role of Controlled Experiments in Software Engineering Research," in *Empirical Software Engineering Issues*, LNCS 4336
- Basili, VR., Selby, RW., Hutchens, DH. (1986). Experimentation in software engineering. *IEEE Trans. Softw. Eng.* 12 (1986), pp. 733–743
- Basili, VR. (1996). The Role of Experimentation in Software Engineering: Past, Current, and Future. *ICSE 18*, 1996, pp. 442–449
- Basili, VR., Rombach, HD. (1988). The TAME project: Towards improvement-oriented software environments, *IEEE Trans. on Software Engineering* 14(6), pp 758-773
- Basili, VR, Caldiera, G., Rombach. D. (1994). Experience Factory. *Encyclopaedia of Software Engineering*, volume 1, pp. 469-476. John Wiley & Sons
- Basili, VR., Caldiera, G. (1995). Improve Software Quality by Reusing Knowledge and Experience. *Sloan Management Review*, 37(1): 55-64
- Basili, VR., Shull, F., Lanubile, F., (1999). Building Knowledge through Families of Experiments, *IEEE Trans. Softw. Eng.* 25 (1999), pp. 456–473
- Basili, VR. (1981). A controlled experiment quantitatively comparing software development approaches, *IEEE Trans Soft Eng SE-7*(3)
- Basili, VR. (1980). *Tutorial on Models and Metrics for Software Management and Engineering*, IEEE Comp Society Press (cat no EHO-167-7), New York
- Basili VR., Reiter, R. (1979). Evaluating Automatable Measures of Software Models, *IEEE Workshop on Quantitative Software Models*, Kiamesha, New York, 1979, pp. 107 - 116.
- Basili, VR., Hutchens, D. (1983). An Empirical Study of a Syntactic Complexity Family, *IEEE Trans. on Soft. Eng.*, Vol. SE-9, No. 6, Nov 1983, pp. 663 - 672.
- Basili, VR. et al.. (1983). Metric Analysis and Data Validation Across FORTRAN Projects, *IEEE Trans. on Soft. Eng.*, Vol. SE-9, No. 6, Nov 1983, pp. 652 - 663.
- Basili, VR., Weiss, DM. (1984). A Methodology for Collection Valid Software Engineering Data. *IEEE Trans. on Soft. Eng.*, 10(11): 728-138, Nov 1984.
- Basili, VR., Briand, L., Melo, WL. (1996). A Validation of Object-Oriented Design Metrics as Quality Indicators, *IEEE Trans. on Soft. Eng.* Oct 1996, 751–761
- Basili, VR., Heidrich, J., Lindvall, M., Münch, J., Regardie, M. (2007). *GQM<sup>+</sup> Strategies - Aligning Business Strategies with Software Measurement*. *ESEM 2007: 488-490*
- Bassman, M. et al. (1995). *Software Measurement Guidebook*, Software Engineering Laboratory Series, Rev. 1, pp. 21-46
- Bayley, I, Zhu. H. (2007). Formalising Design Patterns in Predicate Logic. *SEFM 2007: 25-36*
- Beck, K. (2000). *Extreme Programming Explained*. Addison Wesley.
- Beck, K., Coplien, JO., Crocker, R., Dominick, L., Meszaros, G., Paulisch, F., Vlissides, J. (1996). *Industrial Experience with Design Patterns*, Proceedings of the 18th ICSE, IEEE Computer Society Press

## Reference and Bibliography

- Benediktsson, O., Dalcher, D., Thorbergsson, H., (2006). Comparison of Software Development Life Cycles: A Multi-project Experiment, *IEE Proc.-Softw.*, Vol. 153, No. 3, June 2006, pp 87-101
- Bergner, K. et al. (1998). A Componentware Development Methodology based on Process Patterns. In Joseph Yoder, editor, *Proc. 5th Annual Conf. on the Pattern Languages of Programs (PLoP)*
- Bieman, J. et al. (2003). Design Patterns and Change Proneness, *Proceedings of the IEEE-CS 9th International Software Metrics Symposium (Metrics 2003)*
- Black, TR. (1999). *Doing quantitative research in social sciences: an integrated approach to research design, measurement and statistics*, Sage Publications
- Blaikie, NWH. (2003). *Analyzing quantitative data from description to explanation*, Sage Publications, London
- Blaine, JD., Cleland-Huang, J. (2008). *Software Quality Requirements: How to Balance Competing Priorities*, *IEEE Software* Mar/April 08
- Boehm, B., Basili, VR. (2001). Software Defect Reduction Top 10 List, *IEEE Computer*, Jan 01
- Boehm, BW. (1981). *Software Engineering Economics*. Prentice Hall
- Boehm, B., Brown, JR., Lipow, M. (1976). Quantitative Evaluation of Software Quality. *Proc. 2nd Intl. Conf. on Software Engineering*. Long Beach, Calif.: IEEE Computer Society, Oct. 592-605
- Boehm, B. et al. (1978). *Characteristics of Software Quality*, North Holland Publishing Co. New York
- Booch, G. (2008). *Handbook of software architecture* ([www.booch.com/architecture](http://www.booch.com/architecture))
- Bowling, A. (2002). *Research methods in health* (2<sup>nd</sup> Ed), Open University Press
- Brendan, G. et al. (1996). Social Patterns in Productive Software Organizations. *Annals of Software Engineering*, 259-286. Baltzer Science Publishers, Amsterdam
- Briand, L., Bunse, C., Daly, J. (2001). A Controlled Experiment for Evaluating Quality Guidelines on The Maintainability of Object-Oriented Designs. *IEEE Trans. On Softw. Eng.*, 20 01, 27(6), pp513-530
- Briand, L., Bunse, C., Daly, J. (1997). An Experimental Comparison of the Maintainability of Object-Oriented and Structural Design Documents. *Empirical Software. Engineering*
- Briand, L., Differding, C., Rombach, HD. (1996). Practical Guidelines for Measurement-Based Process Improvement, *Software Process Improvement and Practice Journal*
- Briand, L. (1998). Object Oriented Software Environments, *IEEE Trans. Softw Eng.*, 14 (6)
- Briand, L, Morasca, S., Basili, VR. (1999). Defining and Validating Measures for Object-Based High-Level Design, *IEEE Trans. Software Eng.*, vol. 25, no. 5, pp. 722-741, Sept./Oct. 1999
- Briand, LR., Ermam, K., Morasca, S. (1996). On the Application of Measurement Theory in Software Engineering, *Empirical Software Engineering Journal*, 1(1): 61-88
- Brooks, FP. (1975). *The Mythical Man-Month*, Benjamin/Cummings
- Brooks, FP. (1995). *The Mythical Man-Month: Essays on Software Engineering*, Addison-Wesley
- Brown, WJ. (2000). *Anti-Patterns in Project Management*, Wiley
- Brown, WJ. et al. (1998). *Anti-Patterns: Refactoring Software, Architectures, and Projects in Crisis*, Wiley
- Budgen, D. (2003). *Software Design* (2<sup>nd</sup> Ed). Boston: Addison-Wesley

## Reference and Bibliography

- Budgen, D., Kitchenham, B. et al. (2005). International workshop on realising evidence-based software engineering. ICSE 2005: 687
- Buschmann, F., Meunier, R., Rohnert, H., Sommerlad, P. (1996). Pattern-Oriented Software Architecture, Volume 1, A System of Patterns. Wiley
- Buschmann, F., Henney, K., Schmidt, D. (2007). Past, present, and future trends in software patterns, IEEE Software July/Aug 2007.
- Buschmann, F., Henney, K., Schmidt, D. (2007b). Pattern-Oriented Software Architecture, Volume 2, Patterns for Concurrent and Networked Objects. John Wiley & Sons Ltd
- Campbell, N. (1928), An Account of the Principles of Measurement and Calculation. London: Longmans Green
- Campbell, D., Stanley, J. (1963). Experimental and Quasi-experimental design for research, Chicago: Rand McNally
- Campbell, D., Fiske, D (1959). Convergent and discriminate validation by the multi-trait-multi-method matrix. Psychological Bulletin (56) pp 81-105
- Canfora, G. et al. (2005). A family of experiments to validate metrics for software process models , Journal of Systems and Software, Volume 77, Issue 2, 1 Aug 2005, pp. 113-129
- Carey, J., Carlson, B. (2002) Framework Process Patterns: lessons learned developing application, Addison Wesley
- Carroll, JM. (2000). Scenario-based design of human-computer interactions. MIT Press 2000
- Carver, J., Jaccheri, L., Morasca, S., Shull, F. (2003). Issues in Using Students in Empirical Studies in Software Engineering Education, Ninth International Software Metrics Symposium (METRICS'03)
- Cass, AG. et al. (2000). Little-JIL/Juliette: A Process Definition Language and Interpreter. ICSE 2000
- Chang, W. (2005). Impartial evaluation in software reliability practice, Article Journal of Systems and Software. Volume 76, Issue 2, 1 May 2005, pp. 99-110
- Chidamber, SR., Kemerer, CF. (1994). A metrics suite for object oriented design, IEEE Trans Software Eng, 20 (6), 476-498
- Christensen, LB, (2006). Experimental Methodology (10<sup>th</sup> Ed), Allyn and Bacon inc
- Chrcher, N., Sheppard, M (1995) "towards a conceptual framework for object oriented software metrics, ACM SIGSOFT Software Engineering Notes, Vol 20, No. 2, April 1995
- CIO 2003, The CIO news letter, [www.cio.com/research/itvalue/case.html](http://www.cio.com/research/itvalue/case.html)
- Coad, P., (1992). Object-Oriented Patterns. Com. of the ACM, 1992. 35(9): p. 152-159.
- Cockburn, A. (1996). Prioritising Forces in Software Design, PLoP, 96
- Cockburn, A. (2002). Agile Software Development. Addison-Wesley
- Conte, SD., Dunsmore, HE., Shen, VY. (1986). Software Engineering Metrics and Models, Benjamin Cummings
- Coplien, JO. (2006). Organizational Patterns: Beyond Technology to People. In Enterprise Information Systems VI, Dordrecht, Netherlands, Springer, 2006, pp. 43—52
- Coplien, JO. (1991). Advanced C++ Programming Styles and Idioms. Addison-Wesley, 1991

## Reference and Bibliography

- Coplien, JO., Harrison, N. (2004). *Organizational patterns of agile software development*, Prentice Hall
- Coplien, JO. et al. (2005). *Organizational Patterns: Building on the Agile Pattern Foundations*. Cutter Consortium, Agile Project Management Report 6(6), June 1, 2005
- Coplien, JO., Neil Harrison. (2005). *Organizational Patterns of Agile Software Development*. Upper Saddle River, NJ: Prentice-Hall
- Coplien, JO. (1995). *A Development Process Generative Pattern Language*, *Pattern Languages of Program Design* Addison Wesley, Reading, Mass., 1995. Also in the proceedings of PLoP Conf. 1994
- Coplien, JO., Schmidt, D. (1995). *Pattern Languages of Program Design*, Addison-Wesley Publishing Company
- Coplien, JO., (1996). *Software Patterns*, SIGs Book and Multimedia New York
- Crosby, PB. (1980). *Quality is Free*. McGraw-Hill, London
- Cunningham, W. (1995). *The Checks Pattern Language for Information Integrity in Pattern Languages Program Design*, Addison-Wesley
- Cunningham, W., Back, K. (1987). *Using Pattern Languages for Object-Oriented Programs*, in Proc. OOPSLA'87, Orlando
- Cunningham, W. (1996). *EPISODES: A Pattern Language of Competitive Development*, *Pattern Languages of Program Design 2*, Addison-Wesley Publishing Company
- Dalcher, D. (2003). *Handbook for information systems research*, Idea-Group Publishing
- Dalcher, D. et al. (2005). *Development Life Cycle Management: A Multi-project Experiment*, ECBS'05, IEEE Computer Society Press
- Dalcher, D., (2002). *Life Cycle Design and Management*, in *Project Management Pathways: A Practitioner's Guide*, M. Stevens, Editor. 2002, APM Press: High Wycombe
- Davis, A. et al. (1993). *Identifying and Measuring Quality in a Software Requirements Specification*, IEEE Computer Society Press, Los Alamitos, CA, 1993
- Dekkers, C., Bradley, M. (1999) *It Is the People Who Count in Measurement: The Truth about Measurement Myths*, Crosstalk, The Journal of Defense Engineering, June 1999
- Delano, DE., Rising, L. (1998) *Patterns for System Testing*, *Pattern Languages of Program Design 3*, Addison Wesley Longman, Inc., 1998., pp. 503-525
- DeMarco, T. (1982). *Controlling Software Projects, Management Measurement & estimation*, Prentice Hall
- Deming, WE (1986). *Out of the Crisis*, Cambridge, MA: MIT Press
- Dewberry, C. (2004). *Statistical Methods for Organizational Research: Theory and Practice*. London, Taylor and Francis
- Diaper, D., Stanton, N. (2004). *The handbook of task analysis for human-computer interaction*. NJ: Lawrence Erlbaum Associates.
- Diener, E., Grandall, R. (1978). *Ethics in social and behavioural research*, Chicago, University of Chicago
- Dittmann, T., Gruhn, V., Hagea, M. (2002). *Improved support for the description and usage of process patterns*, 1st Workshop on Process Patterns, OOPSLA 2002
- Dodani, M.H.(2003). *Pattern Driven Software Engineering*, *Journal of Object technology*, Vol 2. No 2., 2003

## Reference and Bibliography

- Dorling, A. (1993). SPICE: Software Process Improvement and Capability dEtermination. *Software Quality Journal* (2), (209)
- D'Souza, DF., Wills, A. (1999). *Objects, Components and Frameworks with UML. The Catalysis Approach.* Addison-Wesley
- Duquenoy, P. et al. (2005\_a). *Social, legal and professional issues of computing*, Middlesex Uni. Press
- Duquenoy, P. (2005\_b) *Ethics of computing in perspectives and policies on ICT in society*, Springer and SBS Media
- Dyson, P., Longshaw, A. (2004). *Architecting Enterprise Solutions: Patterns for High-Capability Internet-Based Systems.* John Wiley & Sons
- Eakin, E (2003). *Architecture's Inscrutable Reformer*, New York Times, July 12, 2003
- Ebert , C. et al. (2005) *Best practices in software measurement*, Springer
- Ebert, C., Dumke, R. (2007). *Software Measurement Establish- Extract-Evaluate-Execute*, Springer
- Eden, AH. (1999). *Precise specification of design patterns and tool support. in their application*, Ph.D. Dissertation, Department of Computer Science, Tel Aviv University
- Eden, D. (2002). *Replication, meta-analysis, scientific progress, and AMJ's publication policy.* *Academy of Management Journal*, 45(5), 841-846.
- Erdogmus, H. (2008a). *The Infamous Ratio Measure*, IEE Software
- Erdogmus, H. (2008b). *Measurement Acquiescence*, IEE Software
- Estabraghy, A., Dalcher, D. (2007a). *A Controlled Experiment to Investigate the Effect of 'Process Patterns' on the Quality of Requirement Analysis*, IEEE AICCSA May 2007
- Estabraghy, A., Dalcher, D. (2007b). *An Investigation of the Impact and Utility of 'Software Patterns' in Software Development Industry*, 20<sup>th</sup> ICSSEA Dec 2007
- Fagan, ME. (1976). *Design and Code Inspections to Reduce Errors in Program Development.* *IBM Systems Journal*, 15(3): 185-211
- Fang, X. (2001): *Using a Coding Standard to Improve Program Quality.* APAQS, pp 73-80
- Fenton N., Ohlsson, N, (2000). *Quantitative Analysis of Faults and Failures in a Complex Software System*, *IEEE Trans. on Soft. Eng.*, 26(8), 797-814, 2000
- Fenton, N., Pfleeger, SL (1991). *Software Metrics: A Rigorous Approach*, Chapman and Hall. NY
- Fenton, N., Neil, M. (1999a), *A Critique of Software Defect Prediction Models*, *IEEE Trans. Software Eng.*, vol. 25, no. 5, pp. 675-689, Sept./Oct. 1999
- Fenton, N., Pfleeger, SL (1997). *Software Metrics: A Rigorous and Practical Approach* (2<sup>nd</sup> Ed). PWS
- Fenton, N. (1994). *Software Measurement: A Necessary Scientific Basis*, *IEEE Trans. on Soft. Eng.*, Vol. 20, No. 3, March 1994, pp. 199 - 206
- Fenton, N., Pfleeger, SL., Glass, RL. (1994). *Science and substance: A challenge to software engineers.* *IEEE Software*, pp. 86-95, July 1994
- Fenton, NE., Melton, A (1996). *Measurement Theory and Software Measurement*, *Software Measurement*, International Thomson Computer Press, 1996, pp. 27-38



## Reference and Bibliography

- Fenton, N., Neil, M. (1999b). Software Metrics: successes, failures, and new direction. *Journal of Systems and Software*, 47, pp. 149-157
- Ferrari, C. (1997). The Road to Maturity Navigating Between Craft and Science, *IEEE Software*, Nov. 77-82.
- Festing, MFW., Altman, D. (2002). Guidelines for the design and statistical analysis of experiments using laboratory animals. *ILAR Journal* 43(4): 244-258
- Field, A. (2000). *Discovering Statistics using SPSS for Windows*, Sage Publications
- Finkelstein, L (1982). Theory and Philosophy of Measurement, in *Theoretical Fundamentals*, vol. 1, *Handbook of Measurement Science*, John Wiley & Sons, 1982, pp. 1-30.
- Florac, WA. et al. (1997). *Practical Software Measurement, Measuring for process management and improvement*, SEI Guidebook, CMU/SEI-97-HB-003
- Florac, WA., Carleton, AD. (1999). *Measuring the Software Process: Statistical Process Control for Software Process Improvement*. Addison-Wesley
- Foote, B. (1994). Lifecycle and Refactoring Patterns That Support Evolution and Reuse. *PLoP 94*
- Foot, B., Yoder, J. (1997). Big Ball of Mud, proceedings of *PLoP 97*
- Foote, B. (1993) A Fractal Model of the Lifecycle of Reusable Objects. *OOPLSA'93*
- Fowler, M. (1997). *Analysis Patterns: Reusable Object Models*. Addison-Wesley.
- Fowler, M. et al. (2002). *Patterns of Enterprise Application Architecture*, Addison-Wesley
- Fuggetta, A. (1998) Applying GQM in an industrial software factory, *ACM Trans. on Soft. Eng. and Methodologies*, Vol. 7, issue 4, pp 411-448
- Furey, S., Kitchenham, B. (1997). Point/counterpoint: function points. *IEEE Software*, 14(2), 63-72
- Futrell, R. et al. (2002) *Quality Software Project Management*, Prentice Hall
- Gabriel, P. <http://hillside.net/patterns/definition.html>
- Gabriel, P., Goldman, R. (2000). *Mob Software: The Erotic Life of Code*, *OOPSLA 2000*
- Gabriel, R. (1996a). Repetition, Generativity, and Patterns, *PLOP Book 2*. Addison-Wesley
- Gabriel, R. (1996b). *Patterns of software*, Oxford University Press
- Gamma, E. et al. (1995). *Design patterns: Elements of Reusable Object-Oriented Software*. Addison Wesley
- Ghezzi, C., Jazayeri, M. (2003). *Fundamentals of Software Engineering (2<sup>nd</sup> Ed)*. Prentice Hall, Upper Saddle River, NJ
- Gillies, A. (1997). *Software Quality: Theory and Management*, International Thomson Computer Press
- Gilb, T. (1988). *Principles of Software Engineering Management*, Addison-Wesley, 1988.
- Gilb, T. (1977), *Software Metrics*, Winthrop Publishers, Inc., Cambridge, Assachusetts
- Gill, GK., Kemerer, CF. (1991). Cyclomatic Complexity Density and Software Maintenance Productivity, *IEEE Trans on Soft Eng*, V 17, No. 12, Dec91, pp. 1284 - 1288.
- Glass, R., Vessey, I., Ramesh, V. (2002). Research in Software Engineering: An Analysis of the Literature, *J. Information and Software Technology*, vol. 44, no. 8, June 2002.

## Reference and Bibliography

- Glass, RL. (1998). *Software Runaways: Lessons Learned from Massive Software Project Failures*. Upper Saddle River, NJ: Prentice-Hall
- Gnats, M. et al. (2001). Towards a living software development process based on process patterns, 8<sup>th</sup> European workshop on software process technology 2001
- Goodman, P. (2004). *Software metrics - best practices for successful IT management*. Philip Jan Rothstein, FBCI. Brookfield, CT, USA, 2004
- Grady, RB., Caswell, DL. (1987). *Software Metrics: Establishing a Company-Wide Program*. Englewood Cliffs, NJ: Prentice-Hall, 1987
- Grady, R. (1992). *Practical Software Metrics for Project Management and Process Improvement*, Prentice Hall, Englewood Cliffs, NJ
- Grady, R. (1994) Successfully Applying Software Metrics, *IEEE Comp*, Vol. 27, No. 9, pp. 18 - 25
- Graham, I. (2003). *A pattern language for web usability*, Addison-Wesley, 2003
- Gueheneuc, Y., Albin-Amiot, H. (2001) Using Design Patterns and Constraints to Automate the Detection and Correction of Inter-Class Design Defects, *Proc. 39th Int'l Conf. and Exhibition Technology of Object-Oriented Langs and Sys*, pp. 296-305, 2001
- Hahsler, M. (2005) A quantitative study of the adoption of design patterns by open source software developers. In S. Koch, editor, *Free/Open Source Software Development*, pp. 103-123. Idea Group
- Hall, T., Fenton, NE. (1997). Implementing effective software metrics programmes, *IEEE Soft.* 14(2), 55-66
- Halstead, MH. (1977). *Elements of Software Science*, New York: Elsevier North Holland
- Hall, T., Fenton, NE. (1997) Implementing effective software metrics programmes, *IEEE Software*, 14(2)
- Hall, T., Baddoo, N., Wilson, D. (2001) *Measurement in software process improvement programmes*, Springer New York pp. 73-82
- Hamming, RW. (1950), Error detecting and error correcting codes. *Bell System Tec. Journal*, 26(2):147-160
- Hardy, GH. (1941). *A Mathematician's Apology* (London 1941).
- Harrison, W., Magel, K. (1981). A topological analysis of the complexity of computer programs with less than three binary branches. *SIGPLAN Not.*, 16(4):51-63, 1981
- Harrison, NB. (1996) *Organizational Patterns for Teams, Pattern Languages of Program Design 2*, Addison-Wesley Publishing Company
- Harrison, N. (1996). Patterns of productive software organizations, *Bell Labs Technical Journal*, 1(1):138-145,
- Hecksel, D. (2004). *Software Development Patterns*, PLOP'04 Conference
- Hefner, K. (1995). An experience-based optimization of the Goal/Question/Paradigm. In *Proceedings of the California Software Symposium*
- Heires, JT. (2001). What I Did Last Summer: A Software Development Benchmarking Case Study, *IEEE Software*, vol. 19, no. 5, Sept/Oct. 2001, p. 33
- Helm, R. (1995). Patterns in practice. *IEEE Trans. on Software Engineering*, 28(6), 595-606
- Hetzel, B. (1993). *Making Software Measurement Work: Building an Effective Measurement Program*, QED Technical Publishing Group, Boston, Massachusetts.

## Reference and Bibliography

- Herbsleb, J. et al. (1997) Software quality and the capability maturity model. *Communications of the ACM* 40(6)
- Hillside group (pattern community website). <http://hillside.net/patterns/>
- Hinkle, M. (2007) Software Quality, Metrics, Process Improvement, and CMMI: An Interview with Dick Fairley, *IT Professional*, vol. 9, no. 3, pp. 47-51, May/June, 2007
- Hitz, M., Montazeri, B. (1996), Chidamber and Kemerer's Metrics Suite: A Measurement Theory Perspective, *IEEE Trans Software Eng.*, vol. 22, no. 4, pp. 267-271, Apr. 1996
- Hoffman, D. (2000), The Darker Side of Metrics, Pacific Northwest Software Quality Conference
- Hone, G. and Stanton, N. (2004) HTA: The development and use of tools for Hierarchical Task Analysis in the Armed Forces and elsewhere, HFI-DTC
- Howell, DC. (2002) *Statistical Methods for Psychology* (5<sup>th</sup> Ed). Duxberry press
- Huang, C., Lo, J., Kuo, S., Lyu, M. (2004). Optimal Allocation of Testing-Resource Considering Cost, Reliability, and Testing-Effort. PRDC 2004: 103-112
- Huang, H., Zhang, S. (2003). Hierarchical process patterns: construct software processes in a stepwise way, *Systems, Man and Cybernetics*, 2003. IEEE International Conference on
- Hughes, B., Cotterell, M. (2005). *Software Project Management*, McGraw-Hill
- Hull, E., Jackson, J., Dick, J. (2005). *Requirements Engineering* (2<sup>nd</sup> Ed). Springer
- Humphrey, W. (1989). *Managing the Software Process*, Reading, MA: Addison-Wesley, 1989
- Hyatt, L. Rosenberg, L., (1996). A Software Quality Model and Metrics for Risk Assessment, *Journal for Software Quality*, Nov. 96
- IEEE Standard 730-2002, IEEE Standard for Software Quality Assurance Plans
- IEEE Standard. 829-1998, IEEE Standard for Software Test Documentation
- IEEE Standard 830-1998, IEEE Standard for Software Requirements Specifications
- IEEE Standard 1028-1997, IEEE Standard for Software Reviews
- IEEE Standard 1012-1998, IEEE Standard for Software Verification and Validation
- IEEE Standard 1061-1998, Standard for a Software Quality Metrics
- Ince, DC. (2003). *Developing Distributed and Ecommerce Applications*, Addison Wesley
- Ince, DC., Sharp, H., Woodman, M. (1993). *Introduction to software project management and quality assurance*, McGraw-Hill
- Ince, DC. (1998). *Software Development: Fashioning the baroque*, Oxford University Press
- Ince, DC. (1991). *Software Quality and Reliability: Tools and Methods*, International Thompson Computer Press
- Ince, DC. (2000). *From data structures to patterns*, Macmillan Press Ltd
- IQPC (2003). The international Quality and Productivity Centre. [www.iqpc.com](http://www.iqpc.com)
- ISO (international Organization for Standardization): ISO 15393. [www.iso.org](http://www.iso.org)

## Reference and Bibliography

- Jacobs, J. (1961). *The death and life of great American cities*. New York, Vintage Books
- Jalil, MJ., Noah, S. (2007) *The Difficulties of Using Design Patterns among Novices: An Exploratory Study*, IEEE Computer Society
- Jeffery, R., Scott, L (2002). *Has twenty-five years of empirical software engineering made a difference?* Software Engineering Conference, 2002. pp. 539 - 546
- Johnson, R., Bhattacharyya, G. (2001). *Statistics: Principles and Methods (4<sup>th</sup> Ed)*, Wiley, New York
- Jones, C. (1986). *Programming Productivity*. New York, NY: McGraw-Hill
- Jones, C. (2007). *Estimating Software Costs (2<sup>nd</sup> Ed)*, McGraw-Hill
- Jones C. (1996). *Applied Software Measurement: assuring productivity and quality (2<sup>nd</sup> Ed)*, McGraw-Hill
- Jung, J. (1971), *The experimenter's dilemma*, New York hopper & Co
- Juristo, N., Moreno, A. (2001), *Basics of Software Engineering Experimentation*. Kluwer Academic
- Kan, SH. (1995). *Metrics and Models in Software Quality Engineering*, Addison-Wesley
- Kampffmeyer, H., Zschaler, S. (2007). *Finding the Pattern You Need: The Design Pattern Intent Ontology*. MoDELS 2007: 211-225
- Kaner, C. Bond, W. (2004) *Software Engineering Metrics: What do they measure and how do we know?* (METRICS '04) Chicago
- Kaplan, RS, Norton, DP. (1996), *The Balanced Scorecard*. Boston: Harvard University Press
- Kaposi, A., Myers, M. (1994). *Systems, models and measures*, Springer-Verlag, London
- Karacan, O. (2000). *Organisational Patterns*, EuroPLOP Conf. Proceedings, EuroPLOP 2000
- Keller, A., Ludwig, H. (2002). *Defining and monitoring service level agreements for dynamic e-business*. Conference Proceedings (LISA 2002), Philadelphia, USA
- Kennedy, J., Bush, AJ. (1984). *An introduction to the design and analysis of experiments*. Lanham, MD: University Press of America, Inc.
- Kerth, N. (1995). *Caterpillar's fate: A pattern language for transformation from analysis to design*, in *Pattern Languages of Program Design*, Addison-Wesley
- Khazanchi, D. Munkvold, BE. (2003). *On the rhetoric and relevance of IS research paradigms: a conceptual framework and some propositions*. 36th Hawaii International Conference on System Sciences
- Khoshgoftaar, TM. et al. (2005). *Resource-oriented software quality classification models*, *Journal of Systems and Software*, Volume 76, Issue 2, 1 May 2005, pp. 111-126
- Kimble, C., Selby, W. (2000). *An interdisciplinary study of information systems: Christopher Alexander and IS failure* (Proc. UKAIS, p256-265)
- Kircher, M., Völter, M. (2007) *Software Patterns*, IEEE Software, July-Aug. 2007
- Kitchenham, B., Pfleeger, SL., Pickard, L., Jones, P., Hoaglin, DC., Emam, K., Rosenberg, J. (2002). *Preliminary guidelines for empirical research in software engineering*. IEEE Trans. on Soft. Eng. 28 (8), 721-734. Aug. 02
- Kitchenham, B., Dybå, T., Jørgensen, M. (2004). *Evidence-Based Software Engineering*. ICSE 2004: 273-281

## Reference and Bibliography

- Kitchenham, B., Mendes, E. (2004). Software Productivity Measurement Using Multiple Size Measures. *IEEE Trans. Software Eng.* 30(12): 1023-1035
- Kitchenham, B. (1987). Towards a constructive quality model, *Soft Eng. Journal*, July 87. pp. 105-113.
- Kitchenham, B., Pfleeger, SL. (1996). Software Quality: The Elusive Target, *IEEE Software*, Jan 1996, pp 12-21
- Kitchenham, B., Pfleeger, SL., Fenton, N. (1995). Towards a Framework for Software Measurement Validation, *IEEE Trans on Soft Eng*, vol.21, No. 12, pp. 929 to 943, Dec95
- Kitchenham, B., Colin, D. (2007) Misleading Metrics and Unsound Analyses *IEEE Software* Mar/Apr 2007
- Kohn, W. (2002). The lost prophet of architecture, *Wilson Quarterly* Summer 2002
- Kompass , <http://www.kompass.co.uk>
- Koziolek, H. (2005). The Role of Experimentation in Software Engineering. Seminar Research Methods, Carl von Ossietzky University of Oldenburg
- Kriz, J. (1988) Facts and Artefacts in Social Science: An Epistemological and Methodological Analysis of Empirical Social Science. McGraw Hill Research
- Kroeber, AL. (1948). *Anthropology: Culture, Patterns and Process*. Harcourt, Brace and World
- Kutz, M., et al. (2003). *Kennzahlen in der IT*. Dpunkt-verlag, Heidelberg, Germany
- Lang, T., Secic, M. (1997). *How to Report Statistics in Medicine: Annotated Guidelines for Authors, Editors and Reviewers*, American College of Physicians, 1997
- Landsberger, H. (1958). *Hawthorne Revisited*, Ithaca
- Laplante, P., Niel, C (2006). *Anti-patterns: Identification, Refactoring, and Management*, CRC Press 2006
- Laplante, P. (2007) *What every engineer should know about software engineering*, CRC Press
- Larman, C., (2004) *Agile and Iterative Development: A Manager's Guide*. Addison-Wesley.
- Larman, C., Basili, VR. (2003) *Iterative and Incremental Development: A Brief History*. *IEEE Computer*, 2003. 36(6): p. 47-56.
- Larman, C. (2002). *Applying UML and Patterns: An Introduction to Object-Oriented Analysis and Design and the Unified Process (2<sup>nd</sup> Ed)*. Prentice-Hall
- Lawler, J., Kitchenham, B. (2003) *Measurement Modelling Technology*, *IEEE Software* Jun 03
- Leedy, PD., Ormrod, JE. (2005). *Practical research: Planning and design (8<sup>th</sup> Ed.)*. New Jersey: Pearson Merrill Prentice Hall
- Lehman, MM. (1980). Programs, life cycles, and laws of software evolution. *IEEE Trans. on Soft. Eng.*, 68(9).
- Lehman, MM. (1987). Process Models, Process Programs, programming Support. Proc. 9<sup>th</sup>. Intern. Conf. Software Engineering, IEEE Computer Society 1987.
- Lethbridge, TC. (2000), What Knowledge Is Important to a Software Professional? *IEEE Computer*, Vol. 33, No. 5, pp 44-50
- Lewis, J. et al. (1991). An Empirical Study of the Object-Oriented Paradigm and Software Reuse, *OOPSLA '91*, pp 184 - 196.

## Reference and Bibliography

- Li, W. (1998). Another metric suite for object-oriented programming. *Journal of Systems and Software*, 44:155-162, 1998.
- Linberg, KR. (1999). Software developer perceptions about software project failure: a case study, *Journal of Systems and Software*, Volume 49, Issue 2-3, 1 Dec 1999, pp. 177-192
- Lim, JS.,et al. (2005). An empirical investigation of the impact of the object-oriented paradigm on the maintainability of real-world mission-critical software, *Journal of Systems and software*, Volume 77, Issue 2, 1 August 2005, pp. 131-138
- Lipsey, MW., Wilson, DB. (2001). *Practical meta-analysis*. Thousand Oaks, CA: Sage Pub.
- Lorenz, M., Kidd, J. (1994). *Object-Oriented Software Metrics*, Prentice Hall Publishing, 1994.
- Ma, J., Wang, Y. (2006). A Quantitative Context Model of Software Process Patterns and Its Application Method Quality Software, Sixth International Conference on, Oct. 2006. pp. 243 - 250
- MacCormack, A., Kemerer, CF., Cusumano, M., Crandall. B (2003) Software Development Practices: Exploring the Trade-offs between Productivity and Quality, *IEEE Software*, vol. 20, no. 5, pp. 78-85
- Madachy, RJ. (2008). *Software Process Dynamics*, Wiley-IEEE Press
- McGarry, J. (2001). When It Comes to Measuring Software, Every Project Is Unique *IEEE Software*, vol. 19, no. 5, Sept./Oct. 2001
- Manns, ML. (2002). Introducing patterns to organizations, *EuroPLOP Conference Proceedings*, EuroPLOP 2002
- Manolescu, D., Kozaczynski, W., Miller, A., Hogg, J. (2007), The Growing Divide in the Patterns World, *IEEE Software*, Vol. 24, Issue 4, Jul-Aug 07, pp 61-67
- Marco, A., Buxton, JN. (1987), *The Craft of Software Engineering*. Addison Wesley
- Martin, R. (1994). *Discovering Patterns in existing application*, PloP94
- Martin, J., McClure, C. (1985). *Diagramming Techniques for Analysts and Programmers*. Prentice-Hall, Englewood Cliffs, NJ.
- Marinescu, R., Ratiu, D. (2004). Quantifying the Quality of Object-Oriented Design, *Proceedings of the 11th IEEE Working Conf. on Reverse Engineering (WCRE 2004)*, IEEE Computer Society Press
- Maxwell, JA. (1996). *Qualitative research design: An iterative approach*. CA: Sage Publications.
- Maxwell, SE., Delaney, HD. (1990). *Design experiments and analyzing data: A model comparison perspective*. Belmont, CA: Wadsworth Publishing.
- Maxwell, KD., Forselius, P. (2000) Benchmarking Software-Development Productivity, *IEEE Software*, v.17 n.1, p.80-88, Jan 2000
- McBurney, DH. (2003). *Research methods (6<sup>th</sup> Ed)*, Brooks/Cole
- McCabe, T. (1976). A Software Complexity Measure, *IEEE Trans. Soft Eng SE-2(4)*, 308-320
- McCall, JA. et al. (1977). *Factors in Software Quality*, Tech. Report. RADC-TR-77-369, Rome Air Development Centre, Air Force Systems Command, Griffiss Air Force Base, N. Y.
- McConnell, S. (1996), *Rapid Development: Taming Wild Software Schedules*, M'soft Press
- McConnell, S. (1997). *Software Project Survival Guide*. Microsoft Press

## Reference and Bibliography

- McConnell, S. (1998) Best Practices: The Art, Science, and Engineering of Software Development. IEEE Software 15(1): 118-120 (1998)
- McCrone, J. (2004) New Scientist, Print Edition. March 2004
- Meli, R. (2000). Functional And Technical Software Measurement: Conflict Or Integration ? FESMA-AEMES 2000 Conference Proceedings, Madrid, October 18-20 2000
- Melton, AC., Gustafson, D., Bieman, J., Baker, A. (1990), A Mathematical perspective for software measures research. IEE Software Engineering Journal, 5(5):246-254, 1990
- Meszaros, G., Doble, J. (1997). A Pattern Language for Pattern Writing, Pattern Language of Program Design 3, Addison-Wesley
- Meszaros, G. (2007) Unit Test Patterns: Refactoring TestCode, Addison-Wesley, 2007
- Meta Group (2002). The business of IT Portfolio-Management: Balancing risk, innovation, and ROI. [www.metagroup.com](http://www.metagroup.com)
- Miller, L. (2003). Pattern language, New York Times July 27, 2003
- Mills, E. (1988). Software Metrics, SEI Curriculum Module SEI-CM-12-1.1, Carnegie Mellon University
- Montgomery, DC. (1997). Design and analysis of experiments. New York : Wiley
- Moore, D., McCabe, G (1993). Introduction to the Practice of Statistics. W.H. Freeman and Company, New York, 1993.
- Moore, GC., Benbasat, I. (1991). Development of an Instrument to Measure the Perceptions of Adopting an Information Technology Innovation, IS Research, Sept, 1991, Vol. 2, No 3, pp 192-222.
- Moore, J. (2000). Combining and Adapting Process Patterns for Flexible Workflow, 11th International Workshop on Database and Expert Systems Applications, September 2000
- Morasca, S. (2003). Foundations of a weak measurement-theoretic approach to software measurement. FASE 2003: 200-215
- Morasca, S., Briand, L. (1997), Towards a Theoretical Framework for Measuring Software Attributes, presented at 4th International Software Metrics Symposium (METRICS '97)
- Morasca, S., Briand, L., Basili, VR., Weyuker, E., Zelkowitz, M (1997b) Comments on Towards a Framework for Software Measurement Validation, IEEE Trans on Soft Eng, pp. 187 - 188 Mar 1997
- Mowbray, T., Malveau, R. (1997). CORBA Design patterns. New York: Wiley Computer Publishing
- Moynihan, T. (1996). An Experimental Comparison of Object-Orientation and Functional-Decomposition as Paradigms for Communicating System Functionality to Users. J. Systems Software, 1996. 33(2): p. 163-169
- Munson, JC., Khoshgoftaar, TM. (1992) Dynamic Program Complexity: The Determinants of Performance and Reliability, IEEE Software November, 1992, pp.48-55
- Munson, JC., Khoshgoftaar, TM. (1990) Applications of a Relative Complexity Metric for Software Project Management, Journal of Systems and Software, Vol 12, No. 3, July 1990, pp. 283-293
- Myers, GJ. (1975) Reliable Software through Composite Design, Van N Reinhold New York
- Nance, RE., Arthur, JD. (2002). Managing software quality: A measurement framework for assessment and prediction. Springer, London, 2002
- NASA. Software Assurance Technology Centre SATC. <http://satc.gsfc.nasa.gov>

## Reference and Bibliography

- Noble, J., Biddle, R. (2002). Patterns as signs. In ECOOP Proceedings
- Oates, BJ. (2005). *Researching Information Systems and Computing*, Sage, 2005
- Odell, JJ. (1998). *Advanced Object-Oriented Analysis & Design using UML*, Sigs Ref. Lib
- Olague, HM. et al. (2007). Empirical Validation of Three Software Metrics Suites to Predict Fault-Proneness of Object-Oriented Classes, *IEEE Trans. Software Engineering* Vol 33. No 6.
- Olsson, T. (2001). V-GQM: A Feed-Back Approach to Validation of a GQM Study, *Metrics '01 International Software Metrics Symposium*, 2001
- Oman, P, Pfleeger, SL. (1997). *Applying software metrics* IEEE Computer Society Press, CA
- Ormerod, TC, Shepherd, A. (2004). *Using task analysis for information requirements specification: The Handbook of Task Analysis for Human-Computer Interaction*. Lawrence Erlbaum Associates
- Osterweil, LJ. (1987). Software processes are software too, *proceedings of ICSE 1987*
- Osterweil, LJ. (1997). Software processes are software too, revisited. *Proceedings of ICSE 1997*
- Page, S., Yates, C (1973). Attitude of psychologists towards the experimenter controls in research, *The Canadian psychologist*, 14, 202-207
- Pant, Y. et al. (1996). Generalization of Object-Oriented Components for Reuse: Measurement of Effort and Size Change, *J. Object-Oriented Programming*, vol. 9, pp. 19- 41, 1996
- Park, RE. et al. (1996). *Goal-Driven Software Measurement: A Guidebook (CMU/SEI-96-HB-002, ADA313946)*. Pittsburgh, Pa.: Soft Eng Institute, Carnegie Mellon University, July 1996
- Patterns Central website. <http://www.patternscentral.com>
- Pattern Community Website. <http://www.hillside.net>
- Perlis AF., Sayward, FG., Shaw, M. (1981). *Software Metrics: An Analysis and Evaluation*. Cambridge, Mass.: MIT Press, 1981
- Parsons, HM. (1974) What Happened at Hawthorne? *Science*, vol. 183, no. 8, pp. 922-932, Mar. 1974
- Perry, D. et al. (2000). Empirical studies of software engineering: a roadmap. *ICSE'02* pp 345-355
- Pfanzagl, J. (1971). *Theory of Measurement (2<sup>nd</sup> Ed)*. Wurzburg, Physica-Verlag, 1971.
- Pfleeger, SL., Fenton, N., Page, S. (1994). Evaluating Software Engineering Standards, *IEEE Computer*, Vol. 27, No. 9, September 1994, pp. 71 - 79.
- Pfleeger, SL., Palmer, JD. (1990). Software Estimation for Object Oriented Systems, *Fall International Function Point Users Group Conference*, Texas, October 1-4, 1990, pp. 181 - 196.
- Pfleeger, SL. et al. (1991). A Software Metrics Database: Support for Analysis and Decision-Making, *Proceedings of the Ninth Annual National Conference on Ada Technology*, March 91, pp. 114 - 119.
- Pfleeger, SL. et al. (1997). Status Report on Software Measurement, *IEEE Software*, March/April 1997, 33-43.
- Pfleeger, SL. (1993). 'Lessons Learned in Building a Corporate Metrics Program.', *IEEE Software*, pp. 67-74.
- Pfleeger, SL. (1999). Albert Einstein and Empirical Software Engineering. In: *Computer* 32 (99), 10, pp. 32-38
- PLoP (1994 to 2007). The 1st to 13<sup>th</sup> Conference on Pattern Languages of Programs



## Reference and Bibliography

Porter, R, Calder, PR. (2004). Patterns in Learning to Program - An Experiment, Proceedings of ACE'2004. pp.241-246

Portland Pattern Repository. <http://c2.com/ppr/>

PSM, Practical Software and Systems Measurement, <http://www.psmc.com/>

Prechelt, L., Unger, B., Philippsen, M. Tichy, WF. (2002). Two Controlled Experiments Assessing the Usefulness of Design Pattern Documentation in Program Maintenance. IEEE Trans on Soft Engineering, 28(6):595-606, June 2002

Prechelt, L., Unger, B., Tichy, WF., Brössler, P., Votta, LG. (2001) A Controlled Experiment in Maintenance Comparing Design Patterns to Simpler Solutions, IEEE Trans. on Soft. Eng., vol. 27, no. 12, pp. 1134-1144, Dec. 2001

Pressman, R., Ince, D. (2000), Software Engineering, a practitioner's approach, European Adaptation, McGraw-Hill

Pressman, R. (2005). Software Engineering: A practitioner's approach (6<sup>th</sup> Ed), McGraw-Hill

Ramesh, B., Jarke, M. (2001). Toward Reference Models for Requirements Traceability, IEEE Trans. on Soft. Eng., v.27 n.1, p.58-93, Jan01

Rees, D. (2001). Essential Statistics (4<sup>th</sup> Ed). Chapman and Hall, 2001

Reibing, R. (2001\_a). Assessing the Quality of Object-Oriented Designs. OOPSLA 2001 Proc.

Reibing, R. (2001\_b). The Impact of Pattern Use on Design Quality Position Paper for the OOPSLA 2001 Workshop

Riehle, D., Zullighoven, H. (1996). Understanding and Using Patterns in Software Development. Theory and Practice of Object Systems, Vol. 2(1), 1996, pp. 33-13.

Rising, L. (1998). CRC Handbook of Object Technology, CRC Press, 1998

Rising, L. (1999). Patterns: A Way to Reuse Expertise, IEEE Communications Mag. Vol. 37, No. 4.

Rising, L., Manns, ML. (2004). Fearless change: Patterns for introducing new ideas, Addison Wesley

Roche, JM, (1994). Software Metrics and Measurement Principles. ACM SIGSOFT Software Engineering Notes 19, 1, 1994, 77-85

Rolland, C., Prakash. N. (1993). Reusable Process Chunks. In Proc of 4th International Conference on Database and Expert Systems Applications. DEXA93, Prague Slovakia, September 1993.

Rombach, D. (1991). Practical Benefits of Goal-Oriented Measurement, in: Software Reliability and Metrics, Elsevier Applied Science, 1991

Rosenberg, L. Hyatt, L., (1996). Developing a Successful Metrics Program, STC '96.

Rosenthal, R. (1998). Covert Communication in Classrooms, Clinics, and Courtrooms, Eye on Psi Chi. Vol. 3, No. 1, pp. 18-22

Royce, W. (1970). Managing the Development of Large Software Systems: Concepts and Techniques. Western Electronic Show and Convention (WesCon) August 25-28, 1970, LA. USA

Rubin, HA. (1996). The Top 10 Mistakes in IT Measurement, IT Metrics Strategies, Vol. II, No. 11, November 1996, [www.cutter.com/benchmark/1996toc.html](http://www.cutter.com/benchmark/1996toc.html)

## Reference and Bibliography

- Rudestam, KE., Newton, RR. (2001). *Surviving your dissertation* (2<sup>nd</sup> Ed), Sage Pub. Inc.
- Salingaros, NA. (1999). *Architecture, Patterns, and Mathematics*. Nexus Network Journal Apr 99.
- Salingaros, NA. (2000). *The Structure of Pattern Languages*. *Architectural Research Quarterly* 4:149-161
- Saltelli A. Tarantola S., Campolongo, F. and Ratto, M., (2004), *Sensitivity Analysis in Practice. A Guide to Assessing Scientific Models*, John Wiley & Sons.
- Sapsford, R. (2007). *Survey research* (2<sup>nd</sup> Ed), Sage Publication
- Saunders, WS. (1999). *From taste to judgment*, *Harvard Design Magazine*, Winter-Spring, 1999, number 7
- Saunders, WS. (2002). *A Pattern Language: reviewed*, *Harvard Design Magazine*, Winter-Spring, 2002, no. 16
- Sauro, J., Kindlund, E. (2005): *A method to standardize usability metrics into a single score*, *Proceedings of ACM CHI/HFCS 2005*. pp. 401-409
- Schach, SR. (2005). *Object-oriented and classical software engineering*, McGraw-Hill
- Scanlan, DA. (1989). *Structured Flowcharts Outperform Pseudo code: An Experimental Comparison*. *IEEE Software* 6(5): 28-36 (1989)
- Schmidt, D, Stal, M., Rohnert, H., Buschmann, F. (2000). *Pattern-Oriented Software Architecture: Patterns for Concurrent and Networked Objects*. Wiley, 2000
- Schmidt, CD. (1995). *Using Design Patterns to Develop Reusable Object-Oriented Communication Software*, *Communication of the ACM* 1995
- Schroeder, M. (1999). *A Practical Guide to Object-Oriented Metrics*, *IT Professional*, v.1 n.6, 30-36, Nov 1999
- SDPP (2002). *Proceedings of the 1st Workshop on Software Development Patterns*
- SEL (1995). *Software Measurement Guidebook*. NASA, Goddard Space Flight Center, Software Engineering Lab, SEL-94-102.
- Seaman, CB. (1999). *Qualitative methods in empirical studies of software engineering*, *IEEE Transaction on software engineering*, Vol 25 No. 4 July 1999.
- Selltiz, C. (1959). *Research methods in social relations*, Holt, New York
- Shalloway, A. (2003). *Can patterns be harmful*, *Cutter IT Journal* September 2003
- Shamoo, A. (2002) *Ethics of the Use of Human Subjects in Research*, Garland Science
- Simons, CL., Parmee, IC., Coward, PD. (2003), *35 years on: to what extent has software engineering design*, *IEE Proceedings – Software*, 150 (6)
- Shaughnessy. JJ., Zechmeister EB. (2002). *Research Methods in psychology*, 6<sup>th</sup> Ed, McGraw-Hill
- Shepherd, A. (2001). *Hierarchical task analysis*. New York: Taylor & Francis
- Shepperd, M. (1996). *Foundations of Software Measurement*. Prentice Hall
- Shepperd, M., Ince, DC. (1993). *Derivation and validation of software metrics*, Clarendon Press
- Shneiderman, B., Mayer, R., McKay, D., and Heller, P. (1977). *Experimental investigations of the utility of detailed flowcharts in programming*. *Communications of the ACM* 20, 6(1977), pp. 373-381

## Reference and Bibliography

- Shull, F., Basili, VR. et al. (2004). Knowledge-Sharing Issues in Experimental Software Engineering. *Empirical Software Engineering An International Journal*, 9, n. 1-2, p. 111-137
- Shull, F., Singer, J., Sjøberg, D. (2008) *Guide to Advanced Empirical Software Engineering*, Springer-Verlag
- Siddle, J. (2007). *Creating Software Architecture using Pattern Sequences*, EuroPlop 2007
- Silverman, M. (1974). The experimenter, *Canadian psychologist* 15
- Singleton, J., Straits, C. (1999). *Approaches to social research* (3<sup>rd</sup> Ed). Oxford University Press
- Singer, j., Vinson, NJ. (2002). Ethical Issues in Empirical Studies of Software Engineering, *IEEE Trans. on Soft. Eng.*, vol. 28, no. 12, pp. 1171-1180, Dec., 2002
- Six sigma <http://software.isixsigma.com/library/content/c051207b.asp>
- Sjøberg, DIK., Anda, B., Anisholm, E., Dyba, T. et al. (2002). Conducting Realistic Experiments in Software Engineering, *Proceedings of the 2002 International Symposium on Empirical Software Engineering (ISESE'02)*
- Sjøberg, DIK., Hannay, J., Hansen, O., Kampenes, V., Karahasanovic, Liborg N., Rekdal, A. (2005). A Survey of Controlled Experiments in Software Engineering, *IEEE Trans. on Software Engineering*, Vol. 31, No. 9.
- Sjøberg, D., Dybå, T., Jørgensen, M. (2007). The Future of Empirical Methods in Software Engineering Research, *29th International Conference on Software Engineering (ICSE'07)*
- Software Engineering Institute (SEI) <http://www.sei.cmu.edu/>
- Sommerville, I. (2007). *Software Engineering* (8<sup>th</sup> Ed), Addison-Wesley
- Standish Group, (2007). *Chaos Chronicles Online: Quality*, The Standish Group International, Inc
- Stanton, NA. (2006). Hierarchical task analysis: Developments, applications, and extensions. *Applied Ergonomics* 37, 55-79
- Steen, LA. (1988). The Science of Patterns, *Science* vol. 240 pp. 611-616.
- Stevens, S. (1946). On the theory of scales of measurement. *Science*, 103, 677-680
- Storrie, H. (2003). Making agile processes scalable. In *ProSim 03*, 2003.
- Storrie, H. (2001). Describing Process Patterns with UML. In Ambriola, Vincenzo (Ed.), *Software Process Technology*, 8th Eur. Ws. EWSPT 2001
- Storrie, H. (2000). *Models of Software Architecture*. PhD thesis, Ludwig-Maximilians-Universität München, Institut für Informatik
- Subramanyam, R. et al. (2003), Empirical Analysis of CK Metrics for Object-Oriented Design Complexity: Implications for Software Defects. *IEEE Trans on Soft Eng* 29(2003)4, pp. 297-310
- Taibi, T., Ngo, DCL. (2001). Why and How Should Patterns Be Formalized. *Journal of Object-Oriented Programming (JOOP)*, vol. 14, no 4, 8-9
- Tang, A., Babar, M., Gorton, I., Han, J. (2006). A survey of architecture design rationale. *Journal of Systems and Software* 79(12): 1792-1804
- Thompson, DW. (1917). *On Growth and Form*, Dover Publications (Revised Edition 1992)
- Tichy, WE. (1998). Should Computer Scientists Experiment More? *IEEE Computer*. pp. 32-40.

## Reference and Bibliography

- Tichy, WF., Lukowicz, P., Prechelt, L., Heinz, EA. (1995) Experimental evaluation in computer science: a quantitative study, *Journal of System Software*. 28 (1995), pp. 9–18
- Thiessen, R. (1994). *Mathematics, the Science of Patterns*, AIMS, April 1994
- Torgerson, S. (1958) *Theory and Methods of Scaling*. New York: John Wiley & Sons
- Tully, C. (1998). *Improving software practice: case experiences*; Wiley; Chichester
- Tully, C. et al. (1999). *Software process analysis and improvement*; IEEE Comp society, California PP 51-106.
- Tsantalis, N., Chatzigeorgiou, A., Stephanides, G. (2006). Design Pattern Detection Using Similarity Scoring. *IEEE transaction on software engineering*, Vol. 32, No. 11, November 2006
- Unger, B., Tichy, WF. (2000). Do design patterns improve communication, An experiment with pair design. *Proc. Int'l Workshop empirical studies of software Maintenance*
- Velleman, PF. Wilkinson, L. (1993). Nominal, ordinal, interval, and ratio typologies are misleading. *The American Statistician*, vol. 47 No. 1, 65-72
- Voas, J., Agresti, W. (2004). Software quality from a behavioural perspective, *IT Pro*, 6(4) pp 46-50, 2004.
- Vokac, M. (2004a). Defect Frequency and Design Patterns: An Empirical Study. *IEEE Transaction on Software Engineering*, VOL. 30, NO. 12, Dec 2004
- Vokac, M., Tichy, W. et al. (2004b). A Controlled Experiment Comparing the Maintainability of Programs Designed with and without Design Patterns: A Replication in a Real Programming Environment, *Empirical Software Eng.*, vol. 9, no. 3, pp. 149-195, 2004
- Wakeland, W. et al. (2004). Using design of experiments, sensitivity analysis, and hybrid simulation to evaluate changes to a software development process. *Software Process: Improvement and Practice* 9(2): 107-119 (2004)
- Walonick DS. (1997). *Survival Statistics*, Statpach Inc
- Walton, CE., Felix, CP. (1977). A Method of Programming Measurement and Estimation, *IBM Systems J.* 16(1), pp 54-65
- Webster, B. (1995). *Pitfalls of Object-Oriented Development*, M&T Books, 1995
- Wegner, P. (1976). Research Paradigms in Computer Science. *Proceedings of the 2nd International Conf. on Software Engineering*, San Francisco, California, US, pp 322 – 330
- Weir, C. (1998). *Patterns for Designing in Teams*, *Pattern Languages of Program Design 3*, Addison Wesley Longman, Inc., 1998
- Weinberg, G. (1992). *Quality Software Management*, Vol. 1, 'Systems Thinking', Dorset House
- Weinberg, G., Schulman, E. (1974). Goals and Performance in Computer Programming, *Human Factors*, vol. 16, pp. 70-77
- Wendorff, P. (2001). Assessment of Design Patterns during Software Reengineering Lessons Learned from a Large Commercial Project, In *Proceeding of CSMR'2001*. pp. 77–84
- Whitenack, B. (1994). RAPPeL: A Requirements-Analysis-Process Pattern Language. Based on the proceedings of PLoP 1994.
- Whitmire, SA. (1997). *Object Oriented Design. Measurement*. John Wiley & Sons. Inc
- Wieggers K. (1997), *Software Metrics: Ten Traps to Avoid*, *Software Dev*, Vol. 5, No. 10

## Reference and Bibliography

- Wiegers, KE. (1999). *A Software Metrics Primer*, Software Development. July 1999
- Wiedenbeck, S. (1999). Novice comprehension of small programs written in the procedural and object-oriented styles. *International Journal of Human-Computer Studies*. 51 (1) (99), pp. 71-87
- Wilson, WM, Rosenberg, LH, Hyatt, LE (1997). Automated Analysis of Requirement Specifications, Nineteenth International Conference on Software Engineering (ICSE-1997)
- Wilson, W. (1999) Writing Effective Natural Language Requirements Specifications, *Crosstalk: The Journal of Defence Software Engineering*, Feb 99.
- Winer, BJ., Brown, DR., Michels, KM. (1991). *Statistical principles in experimental design*. New York: McGraw-Hill, Inc.
- Winn, T., Calder, PR. (2002). Is This a Pattern? *IEEE Software* 19(1): 59-66
- Withall, S. (2007). *Software requirements patterns*, Microsoft press
- Wohlin, C. et al. (2000) *Experimentation in Software Engineering An Introduction*, Kluwer Academic
- Yourdon, E. (2008) Moving beyond SEI-CMM level one, *Software Best Practice Conference*, March 2008
- Yu, TJ. et al. (1988). An Analysis of Several Software Defect Models, *IEEE Trans Soft. Eng.*, Vol 14, No 9, 1988, pp 1261-1270
- Zdun, U. (2007). Systematic pattern selection using pattern language grammars and design space analysis. *Softw., Pract. Exper.* 37(9): 983-1016
- Zelkowitz, MV., Wallace, DR. (1997). Experimental Validation in Software Engineering. In *Information and Software Technology* 39 (1997), pp. 735-743
- Zelkowitz, MV., Wallace, DR. (1998). Experimental models for validating technology. *IEEE Computer*. 23-31
- Zhou, Y. (2006), Empirical Analysis of Object-Oriented Design Metrics for Predicting High and Low Severity Faults. *IEEE Trans. Software Eng.* 32(10): 771-789
- Zuse, H. (1998). *A Framework of Software Measurement*. Berlin, Walter de Gruyter
- Zuse, H. (1991). *Software Complexity: Measures and Methods*, De Gruyter. Berlin

## Appendix A. Experiment Details

In this section the following items are presented:

- Snapshots of online measurement form
- The complete measurement form
- Official marking scheme/criteria for group and individual projects
- Group project assignment
- American Psychological Association Code of Ethics

### Snapshots of online measurement form

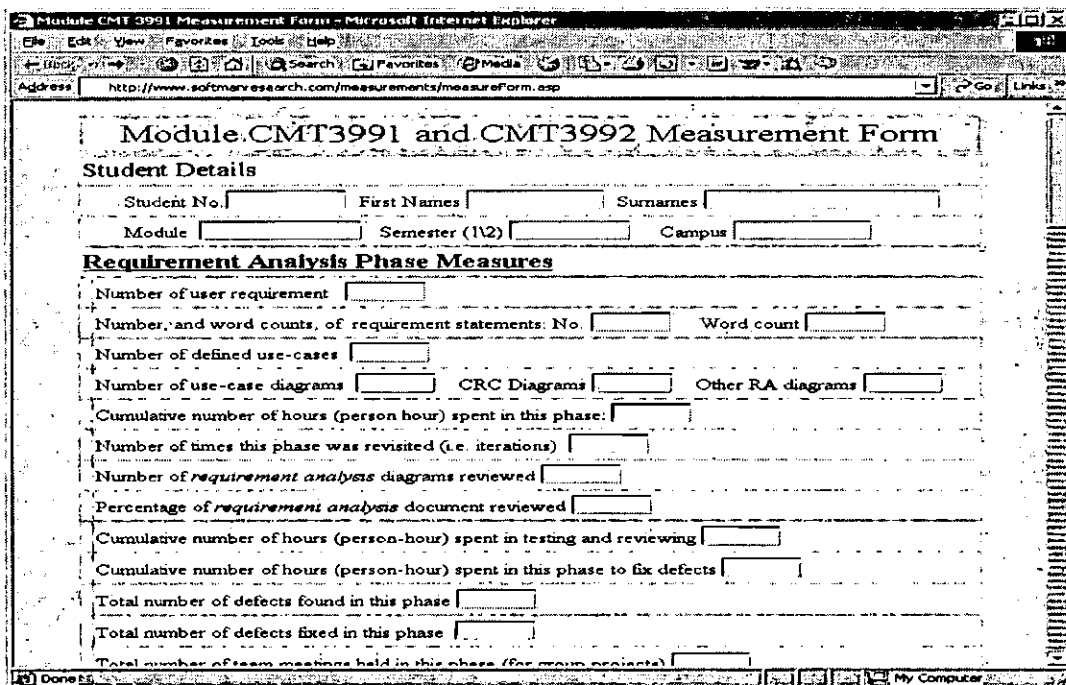


Figure App\_A 1 Snapshots of online measurement form

### The complete measurement form

Module CMT3991 and CMT3992 Measurement Form			
<b>Student Details</b>			
Student No.	First Names	Surnames	
Module	Semester (1/2)	Campus	
<b>Requirement Analysis Phase Measures</b>			
Number of user requirement			

Number, and word counts, of requirement statements: No.  Word count

Number of defined use-cases

Number of use-case diagrams  CRC Diagrams  Other RA diagrams

Cumulative number of hours (person hour) spent in this phase:

Number of times this phase was revisited (i.e. iterations)

Number of *requirement analysis* diagrams reviewed

Percentage of *requirement analysis* document reviewed

Cumulative number of hours (person-hour) spent in testing and reviewing

Cumulative number of hours (person-hour) spent in this phase to fix defects

Total number of defects found in this phase

Total number of defects fixed in this phase

Total number of team meetings held in this phase (for group projects)

Total number of meetings held with project supervisor (for individual projects)

Other info and comments relevant to this phase:

**Design Phase Measures**

Number of class diagrams

Number of activity diagrams

Number of sequence diagrams

Number of other Design diagrams  Name, if any.

Number of user interface (UI) screens

Number of database tables

Number of database relationship diagrams

Number of words in the design document

Number of times this phase was revisited (i.e. iterations)

Percentage of design document reviewed

Number of design models reviewed in this phase

Total number of defects found in this phase

Total number of defects fixed

Cumulative number of hours (person hour) spent in the design phase:

Cumulative number of hours (person hour) spent in the testing the designs

Cumulative number of hours (person hour) spent in this phase on rework to fix defects

Total number of team meetings held in this phase (for group projects)

Total number of meetings held with project supervisor (for individual projects)

Other info and comments relevant to this phase:

**Implementation/Programming Phase**

What programming language(s) was/were used for implementation:

Was a coding standard used  If yes what

Total number of classes ( modules, in the case of non-OOP) developed

Total number of methods (functions, in the case of non-OOP) developed

Total number of developed source lines of code (SLOC)

Total number of inline comments

Number of database queries

Cumulative number of hours (person hour) spent in testing

Cumulative number of hours (person hour) spent in this phase

Number of times this phase was revisited (i.e. iterations)

Number of test cases developed

Number of test cases executed

Percentage of code inspected/reviewed

Total number of defects found in this phase

Total number of defects fixed

Cumulative number of hours spent (person hour) in this phase on rework to fix defects

Total number of team meetings held in this phase (for group projects)

Total number of meetings held with project supervisor (for individual projects)

Other info and comments relevant to this phase:



**Delivery Phase Measures**

Number of test cases developed for the application

Number of test cases executed for the application

Total number of defects found in the application

Total number of defects fixed

Cumulative number of hours (person hour) spent in this phase:

Cumulative number of hours spent (person hour) in testing

Cumulative number of hours spent (person hour) in this phase on rework to fix defects

Total number of team meetings held in this phase (for group projects)

Total number of meetings held with project supervisor (for individual projects)

Other info and comments relevant to this phase:

### Official Marking Scheme/Criteria

Group and individual projects are marked by respective tutors on the 12 marking components depicted in Table App\_A 1. Grading levels and their requirements for the results, design and analysis, evaluation and product criteria are depicted in Table App\_A 2, Table App\_A 3, Table App\_A 4, and Table App\_A 5 respectively.

Abstract
Introduction
Problem Definition
Analysis, Design & Method
Results/Product
Evaluation (of both process & results)
Conclusion (section)
Use & Citation of literature
Research & Concepts
Presentation
Student Competence
Student Management of the Project

Table App\_A 1 Official marking criteria for group and individual projects

Marking Scheme for <i>product</i> Criteria				
Fail	Poor	Average	Good	Excellent
No significant results. OR Product is not working or does not appear to match any part of the original criteria without any attempt at discussion or justification.	Similar findings (or products) are widely available. Distinctive aspects of the problem not covered.	Weaknesses detract seriously but are acknowledged. Results, or product, incorporate adequate testing or validation activities	High standard reached in documentation, software and/or methodology. Findings fit the problem studied and alternatives are compared. OR Software fails gracefully. Remaining errors are acknowledged. Clear indication of assumptions and crucial trade-off decisions.	Results/product offer a notable original feature, quality or purpose. Pathway indicated for further development. Findings are original and could be applied in other projects and appear superior to the usual alternatives OR software is release quality OR output of particular quality presented. Documentation includes advanced issues. Deep understanding of assumptions and trade-offs.

Table App\_A 2 Grading arrangements for *product* criteria

<b>Marking Scheme for Evaluation Criteria</b>				
<b>Fail</b>	<b>Poor</b>	<b>Average</b>	<b>Good</b>	<b>Excellent</b>
No attempt at evaluation. No recommendations stated. No clear idea of how, or if, the recommendations could be implemented.	Lacks objectivity. Only minor relevant evaluation of the work is presented. Limited evaluation without clear links to the objectives.	Some evaluation with some links to work undertaken. Many key issues identified.	Significant evaluation of the outcome (or product) with little emphasis on the process and methods. Clearly stated evaluation firmly based on evidence provided. Feasible set of recommendations linked with project objectives.	Reflective and insightful evaluation of the project and associated conclusions. Assessment of both process and outcome. Choices of approaches and methods re-visited in light of outcomes. Objectives fully reviewed. Clear understanding of potential and limitations. Appropriate and realistic recommendations consistent with results.

Table App\_A 3 Grading arrangements for *evaluation* criteria

<b>Marking Scheme for Design and Analysis Criteria</b>				
<b>Fail</b>	<b>Poor</b>	<b>Average</b>	<b>Good</b>	<b>Excellent</b>
Little attempt at analysis, synthesis and design. Wrong problem addressed.	Major gaps in the analysis and/or design with respect to the original problem	Evidence of analysis and design which appear incomplete in comparison with original problems. Some missing aspects.	Evidence of analysis and design in respect to the original problem.	Analysis and design is explicit. All problems addressed.

Table App\_A 4 Grading arrangements for *Design and Analysis* criteria

<b>Marking Scheme for Project management Criteria</b>				
<b>Fail</b>	<b>Poor</b>	<b>Average</b>	<b>Good</b>	<b>Excellent</b>
Chaos. No planning or organisation. No sign of critical appraisal of project pathway.	A minority of aspects of the project are well managed by the student.	Some aspects of the project are well managed by the student, some managed poorly or not at all.	All aspects of the project are managed by the student, the majority are managed well. Sound planning.	The student evidences self-motivation and self-management throughout the project. High level planning and organisational skills.

Table App\_A 5 Grading arrangements for *Project management* criteria

---

### Project assignment for Group Project (CMT3991)

A private clinic in London called Carex International wants to develop a web application for their intranet system in which patients, doctors, nurses and other relevant healthcare workers can have appropriate access to the system in order to store, retrieve and amend information about the patients. Security and confidentiality are an important aspect of this system and users of the system must have access to a level appropriate to their position and needs. The clinic's system requirements are based on a standard clinic practice, which you are required to collect through investigation. The following is a list of some of the characteristics of the clinic (others will depend on your investigation and assumptions):

- The clinic has 10 senior doctors, 20 junior doctors, 30 nurses and 5 administrators
- The clinic has room for admissions for up to 95 patients.
- Senior doctors can prescribe treatment at all times.
- Junior doctors can only prescribe treatment that has been approved by a senior doctor.
- Junior doctors can discharge patients once approved by a senior doctor.
- Nurses cannot prescribe treatment, but will keep a log of patients' conditions on the system.
- Patients can only read their medical records but cannot modify them.
- Doctors can have access to an up-to-date record of each patient.
- The clinic administrators need to know how many patients are currently admitted and how many are due to be discharged.
- The clinic administrators need to know how many deaths there had been amongst the admitted patients within a period of time.
- Senior doctors need to know how many patients they are responsible for.

#### Tasks:

1. Design and develop a software system for the clinic using an appropriate development lifecycle containing (Requirement analysis, Design, Implementation, and Delivery) phases
  2. Apart from the facts listed above, make any assumptions about the clinic and their requirements that your investigation indicate appropriate and record them. Where possible, support your assumptions or the requirement based on real life data gathered through interviews or other sources.
  3. (For Experimental groups only) Wherever appropriate, use the *process patterns* given to you in the pattern document and record all the instances where they were used.
  4. Produce a report as detailed in the module handbook
  5. Produce a user manual for the software
- 

#### Excerpts from the American Psychological Association Code of Ethics

1. Institutional Approval
2. Informed Consent to Research
3. Informed Consent for Recording Voices and Images in Research
4. Client/Patient, Student, and Subordinate Research Participants
5. Dispensing With Informed Consent for Research
6. Offering Inducements for Research Participation
7. Deception in Research
8. Debriefing
9. Reporting Research Results
10. Plagiarism

---

## Appendix B. Patterns

---

This appendix contains the following sections

- Pattern Philosophy
  - Screenshots from the website hosting process patterns to be used by subjects
  - Sample of Process patterns used for the experiment
  - Examples of general Patterns
- 

### Pattern Theory Philosophy

A basic human impulse is to look for patterns in our surroundings such as time. For example, people organise their daily activities around natural rhythms, such as the rising and setting of the sun.

Although Christopher Alexander is often credited as the founder of the pattern concept with his works in architecture in the 1960's and 70's, the root of the pattern concepts goes back to earlier works in the field of mathematics and natural sciences. Patterns have played a significant part in the field of mathematics and science according to some mathematicians and scientist. Human beings evolved and gained the ability to do mathematics because the mind mimics both natural and man-made patterns [Salingaros 1999]. Hardy [1941] notes 'A mathematician is a maker of patterns'. Steen [1988] also writes, "Mathematics is the science of patterns."

It is also argued [Thiessen 1994] that Kepler's three laws of planetary motions were discovered as a result of his search for patterns of the planetary movements. Furthermore, Newton's formulation of the laws of gravity was also due to his search for patterns in the astronomical data of his day [ibid]. More recently, patterns were defined in the field of anthropology by Kroeber [1948] who introduced the concept of patterns and defined it as follows: "Patterns are those arrangements or systems of internal relationship which give to any culture its coherence or plan, and keep it from being a mere accumulation of random bits. They are therefore of primary importance" [ibid].

However, it was the extensive research work in the field of town and building architecture in the 1960s and 70s that really established patterns as a practical as well as philosophical concept in the field of architecture. Alexander [1977, 1979, 1988] and his colleagues at the Centre for Environmental Structure in Berkeley, California spent more than 20 years developing an approach that was based on a new attitude in architecture and planning which he published in a number of books. Alexander believes that there is a way of building that spans over thousands of years that has always been and will always be valid. He called this the *timeless way*, which is discussed, in the next section.

### Timeless way

At the core of all successful processes of growth, there is one fundamental invariant feature, which is responsible for their success. This way of building has been behind almost all the way of building for thousands of years. The way to identify it, as suggested by Alexander, is to go to a level of analysis, which is deep enough to show what is invariant in all the different versions of this way. This hinges on a form of representation, which reveals all possible construction processes, as versions of one deeper process. Examples of such buildings are traditional villages in Africa, India and Japan as well as religious buildings such as mosques, monasteries of the middle ages, and the temples of Japan. Other examples are the mountain huts of Norway and Austria, cloisters and arcades of English country towns and the cathedral of Pisa.

Alexander [1979] argues that this general deeper process that they all have in common is a quality that cannot be named. He called this the *quality without a name*. This concept is discussed in the next section.

### The quality without a name

In order to seek the timeless way we must first know the *quality without a name*. [Alexander 1979] defines this as follows: "There is a central quality which is the root criterion of life and spirit in a man, a town a building or a wilderness. This quality is objective and precise but cannot be named. The quality cannot be named is not due to

---

its vagueness, but due to a lack of clear, precise and appropriate words to describe it because each word you use to capture it has fuzzy edges and extensions which blurs the central meaning of the quality [ibid]. Therefore, terms, such as 'alive', 'whole', 'comfortable', 'free', 'exact', 'egoless' and 'eternal', used to describe this quality are all insufficient to describe and name this quality.

There is a code, like genetic code, for human acts of building. There is a process that takes place in person's mind when he allows himself to generate building or a place that is alive. Alexander [1979] argued that this process is a language, which he named a *pattern language*. In the next section, this pattern language concept is discussed.

### **Pattern Languages**

People can shape buildings for themselves and have done it for centuries, by using languages, which is called *pattern languages*. A pattern language can give a person who uses it the power to create an infinite variety of new and unique buildings, just as his ordinary language gives him the power to create an infinite variety of sentences. For thousands of years people have used these pattern languages to build houses and towns. In traditional cultures, these processes were common. Even though there are hundreds of farmhouses in the Alps, all similar, yet still each one is beautiful, filled with same elements but in unique combinations so that it is alive and wonderful.

The question, for example, how is a farmer able to make a new barn, lies in the fact that every barn is made of patterns. Although the farmer has some sort of an image of the barn in his mind, this image is not like a drawing or a blue print. It is a system of patterns that function like a language enabling the farmer to make a new barn unlike the ones he made before by combining all the patterns that he knew in a new way. These patterns can be combined and recombined to make an infinite variety of unique barns.

## Screenshots from the website hosting process patterns to be used by subjects

**Login Form** Process Patterns

To be used by authorised user for project assignments at Middlesex University

Student No

First Name

Surname

University Campus:

Module Number:  (CMT3991, Or CMT3992)

Password  ( For help contact: a.estabraghy@mdx.ac.uk )

Figure App\_B 1 Login form

**Pattern Name:** Requirement Analysis Phase

**Problem Definition:** What should be done during the requirement analysis phase?

**Problem Description**

How to start a software development project and lay the foundation is crucially important. Furthermore, how to get the project started and knowing what should be done in this initial phase of the project is essential.

The main goal of the requirement analysis phase is to lay the foundation for a successful project. Unfortunately, however, often the temptation is to ignore or play down the importance of this phase and move on to the so called "the real work" (i.e. the construction phase). This is caused by lack of understanding of the significance of the initiate phase and its critical importance in a successful software development project.

**Context**

As this is the first phase of development, there are not many entry conditions. Here are the two things that should be there before starting:

- There is a requirement for the software.
- There is access to hardware, software and tools required.

**Solution:**

In this phase the project plan should be put in place and initial requirements get defined. The following parallel activities should be taking place in this phase. Note that all three activities must be taking place at the same time.

1. Defining and validating initial requirements. (See Pattern *Stage\_1\_1*).
2. Defining the initial project management (See Patterns *Stage\_2\_1*)
3. Justifying the project (See Pattern *Stage\_3\_1*).
4. Defining the project infrastructure. (See Pattern *Stage\_4\_1*).

Figure App\_B 2 Snapshot of a process patterns hosted online for the experiment

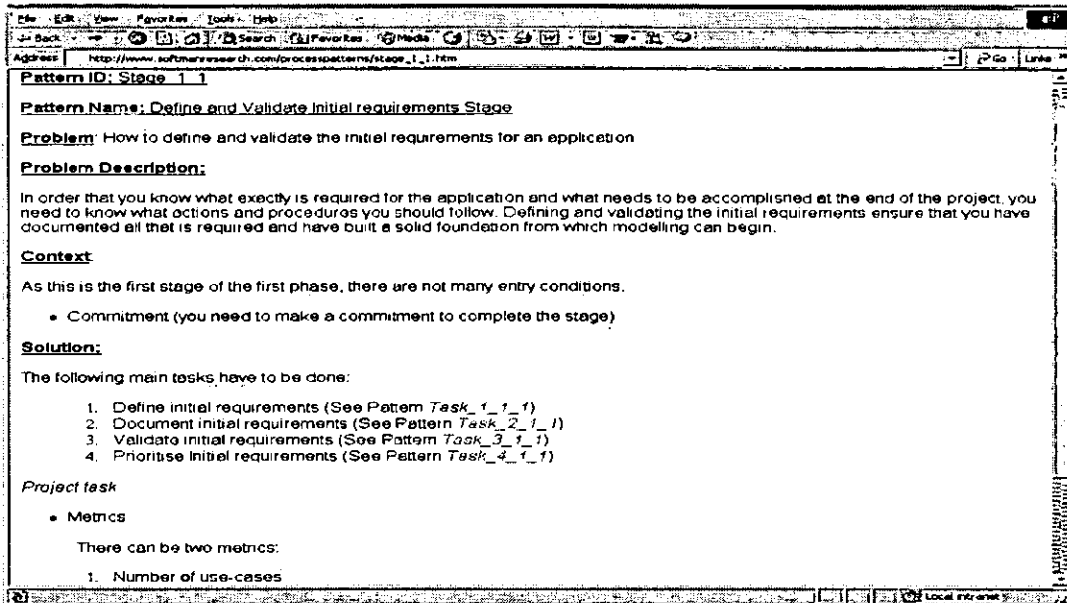


Figure App\_B 3 Snapshot of a process patterns hosted online for the experiment

### A Sample of Process patterns used for the experiment

The following is a shortened sample selection of process patterns, which were used in the experiment as treatment.

**Pattern Name:** Program

**Problem Definition:** How should programming process proceed?

**Solution:**

This involves carrying out a number of tasks that are as follows:

1. Understand the models (See Pattern Task 1 1 3).
2. Reuse existing code and components (See Pattern Task 2 1 3)
3. Document source code (See Pattern Task 3 1 3)
4. Write object oriented source code (See Pattern Task 4 1 3)
5. Synchronise Source code with models (See Pattern Task 5 1 3)
6. Optimise code (See Pattern Task 6 1 3)
7. Create a "build" (See Pattern Task 7 1 3)

**Pattern Name:** Inspect Code

**Problem:** How to inspect code

**Solution:**

Code reviews often reveal problems that normal testing techniques do not; in particular, poor coding practices that make your application difficult to extend and maintain. Code reviews should concentrate on the following issues:

- Making sure that the code satisfies the design
- Naming conventions for your classes, methods and attributes
- Code documentation standards and conventions
- Have you documented what a method does?
- Have you documented what parameters must be passed?
- Have you documented what values are returned by a method?
- Have you documented both what and why a piece code does what it does?



- 
- Writing small methods that do one thing and one thing well
  - Simplifying the code
- 

**Pattern Name:** Test source code

**Problem Definition:** How to carry out testing the source code

**Solution:**

- The solution involves carrying out the following tasks:-
  - Develop/update the master Test/QA plan (See Pattern *Task 1 4 2*)
  - Validate your code (See Pattern *Task 2 4 2*)
  - Record Defects (See Pattern *Task 3 4 2*)
- 

**Pattern Name:** Code testing techniques

**Problem:** How to do code testing

**Solution:**

There are four fundamental code-testing techniques:

- **Black Box testing:** Also called interface testing is a technique in which you create test cases based only on the expected functionality of a method, class or application without any knowledge of its internal workings. The goal of a black box testing is to ensure that the system can do what it should be able to do but not how it does it.
  - **White box testing:** Also referred to as clear box testing or detailed testing, the basic idea in this is that you look at your code and then create test cases that exercise it. The main advantage of white box testing is that it enables you to create tests that will exercise specific lines of code that may not have been tested by simple black box testing
  - **Boundary-value testing:** This is based on the fact that you need to test your code to ensure that it handles unusual and extreme situations. For example in a transaction if someone tried to withdraw -£5.00 or £0.00 the system does not crash and knows how to handle the situation.
  - **Coverage and path testing:** This is a technique in which you create a series of test cases design to test all the code paths in your code. In many ways, this is simply a collection of white box test cases that together exercise every line of code in your application at least once.
- 

**Pattern Name:** Record defects

**Problem:** How to record defects

**Solution**

By recording key information about the defect, you have an accurate description of the problem for repairing it, and you have the data you need to identify weak areas in your software process. It is suggested to record the following information about a defect:

- Description of the defect
  - Date the defect was found
  - Name of the person who found it
  - Defect type
  - Stage the defect was found in
  - Stage that the defect was introduced in
  - Stage that the defect was removed in
  - Date the work was started
  - Date the defect was fixed
  - Steps to recreate the defect
  - Effort, in hours or work days, to fix the defect
  - Description of the solution
- 

**Pattern Name:** Granularity

**Problem:** What should be the components granularity levels?

**Solution:**

- 
- Large methods/classes are more difficult to understand and maintain. Object get things done by collaborating with each other and not by doing everything themselves. This results in smaller classes and shorter methods. If they are large, it is an indication that there is a problem
- 

**Pattern Name:** De-couple Stages**Problem:** How do you de-couple stages (architecture, design, coding) in a development process?**Solution:**

- Link each role to a central role that orchestrates process activities. Parallelism can be re-introduced if the central role pipelines activities.
  - For known and mature domains, serialize the steps. Handoffs between steps should take place via well-defined interfaces. This makes it possible to automate one or more of the steps, or to create a pattern that lets inexperienced staff carry out the step.
- 

**Pattern Name:** Continuity or Seamlessness**Problem:** How to build a system that clearly maps to a model of problem or real world**Solution:**

- Build and integrate user's business model. Build a clear vocabulary of the problem domain
- Cast system requirements in terms of the business model. If the domain model has been clearly defined, system requirements can be discussed and understood more precisely.
- Choose classes based on the business model to maintain traceability, deviations forced by performance, current or planned reuse, and other constraints should be local and clearly documented.
- Maintain development layers (business model to code). Clear separation of domain's system, and technology infrastructure descriptions (and code) helps localise changes.
- Build many projects on the same model

**Related Patterns**

Make a business Model. Construct a System Behaviour Spec

**Pattern Name:** Divide and Conquer**Problem:** How to simplify large implementations**Solution:**

- Construct the implementation to a specification as some form of composition of smaller components. Each design should be constructed in terms of specifications of its parts. There may be many (or many potential) implementations of each component. When you are devising the present implementation, do not consider the internal details of the components. They will have their own decompositions.
- 

**Pattern Name:** Prototype**Problem:** Early acquired requirements are difficult to validate without testing.**Solution:**

- The initial design of a system should focus on the requirements at hand, with broader applicability as a secondary concern. Get something running quickly to obtain design feedback. Build a prototype. Apply techniques such as nouns in the specification imply objects, verbs imply operations, and build on existing objects using inheritance.

**Related Patterns:**

'Application Design is Bounded by Test Design', 'Architect Also Implements', 'Engage Customers', and 'Scenarios Define Problem'.

---

**Pattern Name:** Take No Small Slips**Problem:** How long should the project take?**Solution:**

- Measure how close the critical path (at least) of the schedule is doing. If it is three days beyond schedule, track a 'delusion index' of three days. When the delusion index gets too ludicrous, then slip the schedule. This helps avoid churning the schedule
  - Estimate completion dates using the remaining effort estimates in the work queue report. Calculate each contributor's earliest possible completion date, find the latest of these, and compare that to the hard delivery date for the project. The difference is the completion headroom. The headroom may fluctuate, but steady evaporation of headroom requires management to reorder the work queue, possibly deferring items to a later release date, creating a work split that removes poorly understood or difficult pieces, or holding a recommitment meeting
- 

**Pattern Name:** Process is Product**Problem:** How should a process improvement initiative be organised and managed**Solution**

- Treat it like a development project. Establish a repository to store process documentation and other process artefacts. Use appropriate planning, tracking, configuration management, and other methods and tools, just as they should be used for any other development project. Ensure that the visibility of the project to upper management and the rest of the organisation is comparable to that of other important projects.
- 

**Pattern Name:** Process Follows Practice**Problem:** How do you change the process to meet the required improvement goals?**Solution**

- Start by discovering and understanding current practice throughout the group. Find existing process documentation and talk to practitioners to understand how tasks are performed. Reconcile any differences between actual and espoused processes. Document and review the newly characterised process. Then iteratively and incrementally improve the process and ensure that the documentation is updated appropriately.
- 

**Pattern Name:** Developing in Pairs**Problem:** People are scared to solve problems alone.**Solution:**

- Pair compatible designers to work together; together, they can produce more than the sum of the two individually
- Do not emphasize an individual's special skills. Treat all development as a group activity. This will produce better design decisions and will have a positive effect on the participants. Expertise is shared and everyone in the group learns.
- Allow individuals to create their own short-term work plans. Realize that most of the group activity in a development episode will take place in pairs that find the time to work together. Do not call a meeting to schedule a development episode. Let individuals make their own plans.
- Divide each task into urgent and deferred pieces. (No more than half should be urgent.) Defer more work if necessary to have sufficient headroom. Defer analysis and design for parts that will not be implemented. Both halves of the split should appear in the work queue with different priorities.

**Related Patterns:**Group Validation

---

**Pattern Name:** Requirement Walk-through**Solution:**

- When any member of the work group begins to consider any part of an implied requirement, assemble the entire group. This is a good time to sketch the first informal work plan for that requirement, and it can lead to staffing changes.

- 
- A requirement walk-through will identify relevant information sources, which is retrieved, reviewed, and absorbed as the development episode begins. Collect these information sources as machine-readable examples. Annotate documents so the sources of information will not be lost.
  - Develop a series of well-formatted technical memoranda. Focus each memo on a single subject. Keep it short. Carefully selected, well-written memos can substitute for comprehensive design documentation.
- 

**Pattern Name:** Programming Episode**Solution:**

- Programming should be done in discrete episodes. Select appropriate deliverables for an episode and commit sufficient resources to deliver them. Push for the decisions that can be made. Code the decisions and review the code.
- 

**Pattern Name:** Building the Right Things**Solution:**

- To capture, communicate, and validate software requirements, identify requirements sources. Devise a work plan for interviewing and examining the sources and produce a set of interview results. Capture and validate sponsor objectives as well as manage customer expectations. Prioritise requirements. Establish and keep customer rapport during this process
- 

**Pattern Name:** Defining Requirements**Solution:**

- Create and maintain a glossary of common business terms.
- Use a basic template to specify requirements that organises the information into sections that reflect the activities and types of deliverables needed.
- To verify that behavioural requirements are correct and complete, have all interested parties read the requirements specification. Conduct review meetings. Follow up on all issues raised. Use prototypes. Continue requirements verification through each system development iteration.

**Related Patterns**

Requirements Validation, Behavioural Requirements, Problem Domain Analysis

---

**Pattern Name:** Get Involved Early In Testing**Solution:**

- You are a system tester working on a large software project. To maximise support from the design community, establish a working relationship with the designers early in the project, for example, learn the system and the features along with the designers or attend reviews of requirements and design documentation. Invite designers to reviews of test plans. Do not wait until you need to interact with a designer; by that time it is too late. Trust must be built over time.
  - Start testing when an area is available, but not before. Reach agreement with designers that the area is ready for testing. Agreement is easier if you get involved early.
  - When designers are behind schedule, give them the time they ask for. You will save effort in the long run; testing a poorer-quality system takes more time.
  - Development is drawing to a close. The system is stable. To give a quick evaluation of the overall health of the system, use a favourite killer test to be run at any time. The test should provide good system coverage and be expected to fail, in some manner, most of the time.
- 

**Pattern Name:** Ambiguous Documentation**Solution:**

- To pinpoint possible problem areas, study the documentation. Look for areas that seem ambiguous or poorly defined. If the designers can tell you everything, you need to know about a feature, it probably works. It is what they cannot tell you that needs attention. Get involved early to obtain this information and point it out to designers.

**Related Patterns**

Get Involved Early, Designers Are Our Friends

**Pattern Name:** Scenarios Define Problem**Problem:**

How to define design documents effectively as vehicles to communicate the systems functions

**Solution:**

- Capture system functional requirements as use cases. This defines the problem, and the architecture can proceed in earnest

**Related Patterns:**

Mercenary Analyst

**Pattern Name:** Group Validation**Problem:** How to ensure product quality**Solution:**

- The development team should validate the design.
- Techniques such as CRC cards and group debugging help socialise and solve problems. The CRC design technique has been found to be a great team-builder, and an ideal way to socialise designs. Studies of GBCS projects have found group debugging sessions to be unusually productive.
- Bringing the customer into these sessions can be particularly helpful. The project must be careful to temper interactions between Customer and Developer, using the patterns mentioned in the Resulting Context
- Members of a validation team can also work with QA to fix root causes attributable to common classes of software faults.

**Related Patterns:**

Developing in Pairs

**Pattern Name:** Application Design is Bounded by Test Design**Problem:** When do you design and implement test plans and scripts?**Solution:**

- Scenario-driven test design starts when the customer first agrees to scenario requirements. Test design evolves along with software design, but only in response to customer scenario changes: the source software is inaccessible to the tester. When development decides that architectural interfaces have stabilised, low-level test design and implementation can proceed.

**Related Patterns:**

Engage QA, Scenarios Define Problem

**Pattern Name:** Code Ownership**Problem:** A developer cannot keep up with a constantly changing base of implementation code.**Solution:**

- Each code module in the system is owned by a single developer. Except in exceptional and explicit circumstances, code may be modified only by its owner.
- Lack of code ownership is a major contributor to discovery effort in large-scale software development today. Note that this goes hand-in-hand with architecture: to have ownership, there must be interfaces.
- Arguments against code ownership have been many, but empirical trends uphold its value. Typical concerns include the tendency toward tunnel vision, the implied risk of having only a single individual who understands a given piece of code in-depth, and breakdown of global knowledge.

**Related Patterns:**

Conway's-law, Architect Also Implements, Review the Architecture, Engage Customers, Architect Also Implements, Organisation Follows Market, Interrupts Un-jam Blocking, Review the Architecture

**Pattern Name:** Review the Architecture**Problem:** Blind spots in the architecture and design**Solution:**

- All architectural decisions should be reviewed by all architects. Architects should review each other's code. The reviews should be frequent—even daily—early in the project. Reviews should be informal, with a minimum of paperwork.

**Related Patterns:**

Mercenary Analyst, Code Ownership

**Pattern Name:** Architect Also Implements**Problem:** Preserving the architectural vision through to implementation**Solution:**

- Beyond advising and communicating with developers, Architects should also participate in implementation.

**Pattern Name:** Patron**Problem:** Giving a project continuity**Solution:**

- Give the project access to a visible, high-level manager, who champions the cause of the project. The patron can be the final arbiter for project decisions, which provides a driving force for the organisation to make decisions quickly. The patron is accountable to remove project-level barriers that hinder progress, and is responsible for the organisation's morale (sense of well-being).

**Related Patterns:**

Firewalls, Gatekeeper, Developer Controls Process is in place

**Pattern Name:** Developer Controls Process**Problem:** What role should be the focal point of project communication?**Solution:**

- Place the developer role at a hub of the process for a given feature. A feature is a unit of system functionality (implemented largely in software) that can be separately marketed, and for which customers are willing to pay. The developer is the process information clearinghouse. Responsibilities of developers include understanding requirements, reviewing the solution structure and algorithm with peers, building the implementation, and unit testing.
- Note that other hubs may exist as well.

**Related Patterns:**

Work Flows Inward, Move Responsibilities, Mercenary Analyst, Firewalls, Gatekeeper, and Buffalo Mountain.

**Pattern Name:** Form Follows Function**Problem:** A project lacks well-defined roles**Solution:**

- Group closely related activities (that is, those mutually coupled in their implementation, or which manipulate the same artefacts, or that are semantically related to the same domain). Name the abstractions resulting from the grouped activities, making them into roles. The associated activities become the responsibilities (job description) of the roles.

**Resulting Context:**

Organisation Follows Location, Organisation Follows Market, and Architect Also Implements.

**Pattern Name:** Size the Schedule**Problem:** How long should the project take?**Solution:**

- The external schedule is negotiated with the customer; the internal schedule, with development staff. The internal schedule should be shorter than the external schedule by two or three weeks for a moderate

---

project. If the two schedules cannot be reconciled, either customer needs, or the organisation's resources, or the schedule itself must be re-negotiated.

- Reward developers for meeting the schedule

#### **Related Patterns**

Compensate Success,

---

**Pattern Name:** Self-Selecting Team

**Problem:** How to build teams

**Solution:**

- Build self-selecting teams, doing limited screening on the basis of track record and broad interests. An empowered, enthusiastic team willing to take extraordinary measures to meet project goals
- 

**Pattern Name:** Big Ball of Mud

**Problem:**

Overgrown, tangled, haphazard spaghetti code is hard to comprehend, repair, or extend, and tends to grow even worse if it is not somehow brought under control

**Solution:**

- If you cannot easily make a mess go away, at least cordon it off. This restricts the disorder
  - To a fixed area, keeps it out of sight, and can set the stage for additional re-factoring.
  - If your code has declined to the point where it is beyond repair, or even comprehension, throw it away it and start over
- 

**Pattern Name:** Development Artefacts

**Problem:**

What development artefacts are created, modified, or accessed within each activity?

**Solution:**

Consider evolution as interactions between the management and software development artefacts and the users of the development process. Describe scenario instances as interactions between development artefacts and users of the development process.

---

#### **An example of Anti-Patterns [Coplien 1996]**

**Name:** Egalitarian Compensation

**Problem:** Providing appropriate motivation for success

**Context:** A community of developers meeting night schedules in a high-payoff market.

**Forces:** Disparate rewards motivate those who receive them, but may frustrate their peers. You want to encourage team cohesion, build team identity, and in general encourage team behaviour.

**Supposed solution:** The entire team (social unit) should receive comparable rewards, to avoid de-motivating individuals who might assess their value by their salary relative to their peers.

**Resulting Context:** An organisation where people feel accepted as peers. However, leaders will still emerge and there will still be an inequitable distribution of work; that distribution of work is no longer commensurate with compensation. People figure this out, and lose one of their motivations to excel. The pattern has the opposite effect of encouraging behaviour where people over-extend themselves.

## Appendix C. Metrics Specifications

In this appendix, the specification of the metrics, used in this research, is presented in table formats. The measurements presented here are divided into two groups: direct and, indirect (derived). The derived measurements are presented in 'metric tables' followed by direct measures (named measures) which are presented in 'measure tables'.

The metric selected for this study are based on the GQM model described in Section 4.8.2 and 6.2. As the process patterns used as treatment in the experiment cover a complete development lifecycle, there was a wide range of software metrics that could be used in the experiment. However, due to scope limitation of this project, a limited number of metrics were selected to be analysed and reported. The metrics cover the four major phases of the development lifecycle (i.e. Requirement Analysis, Design, Implementation, and Delivery).

The tables contain a number of elements that are explained in Table APP\_C 1 below.

Element	Description
<b>Definition</b>	A concise definition of the metric
<b>GQM Goal</b>	The metrics were developed using the (Goal/Question/Metric) model to satisfy a quality goal. This element states the GQM goal with which this metric is associated
<b>GQM Question</b>	The associated GQM (Goal/Question/Metric) question. (i.e. the question to which the metric provides answer)
<b>Type</b>	The nature of the metric type. Can be either <i>qualitative</i> , <i>quantitative</i> or both.
<b>Source</b>	The source of metric value (i.e. experiment subject, calculated, or researcher).
<b>Applicable Phase</b>	The development phase to which the metric is applicable
<b>Rationale</b>	A rationale for the metric and the objective to be achieved
<b>Purpose</b>	The purpose and objective of the metric
<b>Related metrics</b>	Other metrics that are related to this metric
<b>Scope</b>	The scopes in which this metric is applied
<b>Evaluation Method</b>	Procedure and method used for metric evaluation
<b>Attribute to measure</b>	The software attribute that the metric is to measure
<b>Measurement Scale</b>	The least applicable measurement scale (i.e. nominal, ordinal, interval, ratio, absolute) for the purpose of statistical analysis
<b>Required Measurement</b>	Other related measures that the metric requires to be evaluated
<b>Metric's Value</b>	The way the metric value is attained

Table APP\_C 1 A description of the elements of the metric specification table

The following tables present metric specifications.



<i>Metric</i>	
<b>Percentage of Traceable Requirements (Requirements Traced per Requirements Defined)</b>	
<i>Description</i>	
<b>Definition</b>	Measures the percentage of the requirements that are traceable (Traceable Requirements per Total Requirements Ratio).
<b>GQM Goal</b>	Requirement Artefact Quality
<b>GQM Question</b>	What percentage of the requirements is traceable?
<b>Type</b>	Quantitative
<b>Evaluation Method</b>	As per formula below
<b>Applicable Phase</b>	Requirement Analysis
<b>Rationale</b>	Requirements traceability refers to the ability to describe and follow the life of a requirement, in both a forwards and backwards direction [Ramesh and Jarke 2001]. A requirement should be linked to a higher level document (i.e. source), which could be a higher-level system requirement, as well as downward to the design elements, source code, and test cases that are constructed to implement and verify the requirement [Hull et al. 2005] [Davis 1993]. Therefore, the higher the rate of traceable requirements in a software project, the higher the quality of the requirements and the Requirement Analysis phase.
<b>Purpose/Objective</b>	This metric was used in the experiment to determine if there was any difference between the treated and control groups in terms of the percentage of traceable requirements. The metric will show whether, as a result of using process patterns, the treated groups will have a higher percentage of their requirements traced to design, test, and implementation.
<b>Scope</b>	Small software development projects
<b>Attribute to Measure</b>	Traceability of Requirements
<b>Metric Scale</b>	Interval
<b>Related Measure(s)</b>	Number of Requirements ( Measure 2), Number of Traceable Requirements (Measure 1)
	<i>Required Measurement</i>
Number of Traceable Requirements (NTR)	<i>Metric's Value</i> $\frac{NTR}{NR} \times 100$
Number of Requirements (NR)	

**Metric 1** Percentage of Traceable Requirements

<i>Metric</i>	
<b>Percentage of Defects Fixed (Defect removal ratio)</b>	
<i>Description</i>	
<b>Definition</b>	The metric measures percentage of defects that were fixed in a development phase.
<b>GQM Goal</b>	Test/Review Quality
<b>GQM Question</b>	What percentage of detected defects is fixed?
<b>Type</b>	Quantitative
<b>Evaluation Method</b>	Calculated as per formula below
<b>Applicable Phase</b>	Requirement analysis, Design, Implementation, and Delivery
<b>Rationale</b>	Defect control and management is important in software development, as defects are a root cause of software failures [Jones 2007]. Therefore, a development process in which more of the defects are fixed is more likely to produce a more reliable software product. This metric was used to provide an indication of the quality of defect correction process by assessing the percentage of the defects that were fixed, for each development phase. A higher value would indicate a better defect correction process as well as a less erroneous product.
<b>Purpose/Objective</b>	This metric was used in the experiment to determine if there was any difference between the treated and control groups in terms of the percentage of the defects fixed. The metric will show whether as a result of using process patterns the treated groups will fix a higher percentage of the defects.
<b>Scope</b>	Small software development projects
<b>Attribute to Measure</b>	Defect correction process
<b>Metric Scale</b>	Interval
<b>Related Measure(s)</b>	No. of Defects Detected (Measure 3), No. of Defects Fixed (Measure 4)
	<i>Required Measurement</i>
No. of Defects Fixed (NDF)	<i>Metric's Value</i> $\frac{NDF}{NDD} \times 100$
No. of Defects Detected (NDD)	

**Metric 2** Percentage of Defects Fixed

<i>Metric</i>	
<b>Percentage of Requirement Specification Document Reviewed</b>	
<i>Description</i>	
<b>Definition</b>	The metric measures the percentage of the requirement specification document (RSD) reviewed.
<b>GQM Goal</b>	Test/Review Quality
<b>GQM Question</b>	What percentage of the requirement specification document is reviewed?
<b>Type</b>	Quantitative
<b>Source</b>	Experiment Subjects
<b>Applicable Phase</b>	Requirement analysis
<b>Rationale</b>	Reviews are the most widely used approach for assessing software quality [Sommerville 2007]. The higher the percentage of the requirements document reviewed the better the quality of the review process and the better the chance of finding any defects [Fagan 1976]. Furthermore, inspection of requirements and design are more effective than testing [Hinkle 2007]. Therefore, a higher value for this metric would indicate a better process as well as a better product in terms of the requirement specification document. However, one difficulty with this metric, as pointed out by Nance and Arthur [2002], is that it does not take the thoroughness of the review into consideration and the focus is on quantity rather than quality. This, however, does not affect the results of the experiment, as the random nature of the treated and control groups means that thoroughness of the reviews is equally spread between treated and control groups.
<b>Purpose/Objective</b>	This metric was used in the experiment determine if there was any difference between the treated and control groups in terms of the percentage of RSD reviewed.
<b>Scope</b>	Small software development projects
<b>Attribute to Measure</b>	Requirement reviews
<b>Metric Scale</b>	Interval
<b>Related Measure(s)</b>	Percentage of Design Document Reviewed (Metric 5), Percentage of Source Code Reviewed (Metric 8)
<i>Required Measurement</i>	
<i>Metric's Value</i>	
Percentage of RSD Reviewed	Percentage of RSD Reviewed

**Metric 3 Percentage of Requirement Specification Document Reviewed**

<i>Metric</i>	
<b>Percentage of Phase Time Spent in Testing</b> (Test Time Per Phase time ratio)	
<i>Description</i>	
<b>Definition</b>	The metric measures the percentage of the development phase time spent in testing.
<b>GQM Goal</b>	Test/Review Quality
<b>GQM Question</b>	What percentage of phase time was spent in testing?
<b>Type</b>	Quantitative
<b>Evaluation Method</b>	Calculated as per formula below
<b>Applicable Phase</b>	Requirement analysis, Design, Implementation, and Delivery
<b>Rationale</b>	A right proportion of the phase time allocating to testing is important in providing the necessary time for carrying out the required testing tasks adequately. A small proportion of the phase time allocated to tests would indicate a deficiency and inadequacy in carrying out the test tasks properly. Normally between 30 to 40 percent of project effort is spent on testing [Pressman and Ince 2000]. It is generally recommended in the literature that in most cases between 30 to 50 percent of the development effort should be allocated to testing [Six sigma] [Huang 2004]. This, however, should be much higher in the case of human-rated applications such as flight control.
<b>Purpose/Objective</b>	This metric was used in the experiment to determine if there were any difference between the treated and control groups in the experiment in terms of the percentage of the phase time spent in testing.
<b>Scope</b>	Small software development projects
<b>Attribute to Measure</b>	Time Spent in Testing
<b>Metric Scale</b>	Interval
<b>Related Measure(s)</b>	Phase Test Time (Measure 7), Phase Time (Measure 5)
<i>Required Measurement</i>	<i>Metric's Value</i>
Phase Test Time (PTT)	$\frac{PTT}{PT} \times 100$
Phase Time (PT)	

**Metric 4** Percentage of Phase Time Spent on Testing

<i>Metric</i>	
<b>Percentage of Design Document Reviewed</b>	
<i>Description</i>	
<b>Definition</b>	The metric measures percentage of the design document reviewed.
<b>GQM Goal</b>	Test/Review Quality
<b>GQM Question</b>	What percentage of the design document was reviewed?
<b>Type</b>	Quantitative
<b>Source</b>	Experiment Subjects
<b>Applicable Phase</b>	Design
<b>Rationale</b>	Inspection of requirements and design are more effective than testing [Hinkle 2007]. The higher the percentage of design document reviewed the better the chance of finding any defects in both the modelling and related artefacts. Therefore, a higher value for this metric would indicate a better process as well as a better product in terms the product design.
<b>Purpose/Objective</b>	This metric was used in the experiment to determine if there was any difference between the treated and control groups in terms of the proportion of design document reviewed.
<b>Scope</b>	Small software development projects
<b>Attribute to Measure</b>	Design Review Quality
<b>Metric Scale</b>	Interval
<b>Related Measure(s)</b>	Percentage of Source Code Reviewed (Metric 8), Percentage of Requirement Specification Document Reviewed (Metric 3)
<i>Required Measurement</i>	<i>Metric's Value</i>
Percentage of Design Document Reviewed	Percentage of Design Document Reviewed

**Metric 5** Percentage of Design Document Reviewed

<i>Metric</i>	
<b>Methods per Class Ratio</b>	
<i>Description</i>	
<b>Definition</b>	The metric measures the average number of methods per class. This is also referred to as WMC (Weighted Methods per Class) in Object Oriented terminology.
<b>GQM Goal</b>	Design Artefacts Quality
<b>GQM Question</b>	How many methods are defined per class?
<b>Type</b>	Quantitative
<b>Source</b>	Experiment Subjects
<b>Applicable Phase</b>	Design
<b>Rationale</b>	An application developed with more finely granular objects (i.e. a lower number of methods per class) is likely to be more easily maintained and reusable as objects should be smaller and less complex [Schroeder 1999]. A larger number of methods per class are likely to hinder extensibility and complicate testing due to the increased object size and complexity. The larger the number of methods, the more complex the inheritance tree and the more limiting the potential reuse. Number of methods per class therefore should be kept as low as possible [Pressman and Ince 2000]. This metric was first proposed by Chidamber and Kemerer [1994], referred to as Weighted Method per Class, as a measure of complexity (see Section 4.9).
<b>Purpose/Objective</b>	This metric was used in the experiment to determine if there was any difference between the treated and control groups in terms of the number of methods per class ratio.
<b>Scope</b>	Small software development projects
<b>Attribute to Measure</b>	Granularity/Complexity/maintainability
<b>Metric Scale</b>	Interval
<b>Related Measure(s)</b>	Number of Classes (Measure 9), Number of Methods (Measure 10)
	<i>Required Measurement</i>
No. of Methods (NOM)	$\frac{\text{NOM}}{\text{NOC}}$
No. of Classes (NOC)	NOC

**Metric 6** Methods per Class Ratio

<i>Metric</i>	
<b>Productivity</b>	
<i>Description</i>	
<b>Definition</b>	The metric measures productivity as 'Rate of output per unit input', where the output is the value delivered and the input is the resources.
<b>GQM Goal</b>	Development Artefacts Quality
<b>GQM Question</b>	What is the productivity of the development phase?
<b>Type</b>	Quantitative
<b>Evaluation Method</b>	As per formula below
<b>Applicable Phase</b>	Implementation
<b>Rationale</b>	<p>Productivity evaluation is difficult and controversial and even advice offered by ISO 15393 on productivity measurements have been shown to be misleading [Kitchenham and Colin 2007]. Difficulties in productivity measurement are partly due to the diverse and differing ways and views on how input and output should be measured and the difficulty in measuring them [Kitchenham and Mendes 2004] [Shepperd 1996] [Walton and Felix 1977]. For example, LOC as a measure of output does not take into account many attributes such as verbosity of the programmer, the programming language, and environmental complexity such as skills, pressure, tool support, computing platform (see Section 4.10). However, LOC and Function Point counts are the most common output measurements used [Maxwell and Forselius 2000]. While some argue that it is unsafe to measure productivity as a ratio of two unrelated variables [Kitchenham and Colin 2007]], productivity as size over effort ratio is by far the most popular method of evaluating productivity. In a literature review of the productivity measurement, Kitchenham and Mendes [2004] found that (with the exception of one) all the surveyed papers to use this method of productivity evaluation. Although imperfect, this method of productivity measurement (i.e. LOC/Effort) is widely used and provides a consistent measure of productivity [MacCormack et al. 2003]. The method of productivity measurement employed in this research is also size over effort, where size is measured in terms of the number of lines of code (LOC), and effort in terms of person-hour. As the focus of the experiment is on the development phase, productivity in the Implementation phase (i.e. time spent in the Implementation phase) is evaluated. The overall productivity has also been evaluated.</p> <p>The method by which LOC is measured, and other related factors and issues (e.g. verbosity of the programmer, the programming language, and environmental complexity such as skills, pressure, tool support, computing platform), are randomly spread amongst the control and treated groups in this study, and have therefore neutralised effect. LOC over Effort is therefore appropriate for this study as a way of comparing treated and control groups in terms of their productivity.</p>
<b>Purpose/Objective</b>	This metric was used in the experiment to determine if there was any difference between the treated and control groups in terms of productivity.
<b>Scope</b>	Small software development projects
<b>Attribute to Measure</b>	Productivity
<b>Metric Scale</b>	Interval
<b>Related Measure(s)</b>	Development Phase Time (Measure 5), LOC (Measure 8), Total development time (Measure 6)
<i>Required Measurement (Implementation Productivity)</i>	
No. of Lines of Code (LOC)	LOC
Implementation Phase Time (IPT)	IPT
<i>Required Measurement (Overall Productivity)</i>	
No. of Lines of Code (LOC)	LOC
Total Development Time (TDT)	TDP

Metric 7 Productivity

<i>Metric</i>	
<b>Percentage of Source Code Reviewed</b>	
<i>Description</i>	
<b>Definition</b>	The metric measures percentage of source code that was reviewed.
<b>GQM Goal</b>	Review Quality
<b>GQM Question</b>	What percentage of source code was reviewed?
<b>Type</b>	Quantitative
<b>Source</b>	Experiment Subjects
<b>Applicable Phase</b>	Implementation
<b>Rationale</b>	An error detected within the development process is from 10 to 100 times less costly to fix than a defect found during the application's operation [Boehm and Basili 2001] [Standish Group 2007]. The higher the percentage of code inspected the better the chance of finding faults and deficiencies in code [Fagan 1976]. Therefore, a higher value for this metric would indicate a better process as well as a better product in terms of the produced code.
<b>Purpose/Objective</b>	The objective of the metric is to determine if there was any difference between the treated and control groups in terms of the percentage of source code reviewed
<b>Scope</b>	Small software development projects
<b>Attribute to Measure</b>	Code review
<b>Metric Scale</b>	Interval
<b>Related Measure(s)</b>	Percentage of Design Document Reviewed (Metric 5), Percentage of Requirement Specification Document Reviewed (Metric 3)
<i>Required Measurement</i>	<i>Metric's Value</i>
Percentage of Source Code Reviewed	Percentage of Source Code Reviewed

Metric 8 Percentage of Source Code Reviewed



<b>Metric</b>	
<b>Defect Density</b>	
<b>Description</b>	
<b>Definition</b>	The metric measures defect density as the ratio of the number of defects to program length (defect/size).
<b>GQM Goal</b>	Test/Review Quality
<b>GQM Question</b>	What is the rate of the defect density?
<b>Type</b>	Quantitative
<b>Evaluation Method</b>	Calculated as per formula below
<b>Applicable Phase</b>	Implementation Phase
<b>Rationale</b>	<p>This metric is generally used in industry for many purposes such as identifying candidate components for further inspection or analysing and tracking the impact of defect removal on quality improvement [Ebert 2005]. A reduction in defect density is important especially as studies have found that up to 65% of defects occur at the design and coding stages [Boehm 1981][Jones 1996]. It is the most commonly used means of measuring quality of a piece of software code and has become the de-facto industry standard measure of software quality [Fenton and Pfleeger 1997]. One criticism of this metric is that it relies on measures (i.e. defects and size) which are difficult to define and measure.</p> <p>This metric was used in this research to provide an indication of the quality of the source code in terms of defects. A lower value would indicate a better quality product (i.e. source code).</p>
<b>Purpose/Objective</b>	This metric was used in the experiment to determine if there was any difference between the treated and control groups in terms of the rate of defect density. The metric will show whether using process patterns by the treated groups reduced defect density.
<b>Scope</b>	Small software development projects
<b>Attribute to Measure</b>	Defect Density
<b>Metric Scale</b>	Interval
<b>Related Measure(s)</b>	No. of Defects Detected (Measure 3), Program Size LOC (Measure 8)
	<b>Metric's Value</b>
No. of Defects Detected (NDD)	NDD
Program Size (LOC)	LOC

Metric 9 Defect Density

<i>Metric</i>	
<b>Comment Density</b>	
<i>Description</i>	
<b>Definition</b>	The metric measures percentage of source code that has been commented.
<b>GQM Goal</b>	Development Process Quality
<b>GQM Question</b>	What percentage of lines of code is commented?
<b>Type</b>	Quantitative
<b>Evaluation Method</b>	As per formula below
<b>Applicable Phase</b>	Implementation
<b>Rationale</b>	The comment density metrics is useful for estimating the quality of the code [Lorenz and Kidd 1994]. The higher the percentage of code that is commented the better the quality of code in terms of readability, modifiability and maintainability. It is generally recommended that there should be as many lines of comments as there lines of code [Ambler 1998].
<b>Purpose/Objective</b>	This metric was used in the experiment to determine if there was any difference between the treated and control groups in terms of the proportion of source code commented.
<b>Scope</b>	Small software development projects
<b>Attribute to Measure</b>	Code readability/clarity
<b>Metric Scale</b>	Interval
<b>Related Measure(s)</b>	Number of Lines of Comment (Measure 11), LOC (Measure 8)
<i>Required Measurement</i>	<i>Metric's Value</i>
No. of Lines of Comments (LOCom)	$\frac{LOCom}{LOC}$
No. of Source Lines of Code (LOC)	LOC

**Metric 10** Comment density

<i>Metric</i>	
<b>Test Case Density (Test case coverage)</b>	
<i>Description</i>	
<b>Definition</b>	The metric measures the extent to which testing covers the applications functionality. This is also referred to as Test Case Coverage.
<b>GQM Goal</b>	Test Quality
<b>GQM Question</b>	What is the test case per requirement ratio
<b>Type</b>	Quantitative
<b>Evaluation Method</b>	As per formula below
<b>Applicable Phase</b>	Delivery
<b>Rationale</b>	This metric provides an indication of the test coverage with respect to requirements. Every requirement should have one or more tests associated with it [Laplante 2007]. A higher Test Case per Requirement Ratio denotes a more thorough and comprehensive test process as it offers a higher probability of detecting any defects.
<b>Purpose/Objective</b>	This metric was used in the experiment to determine if there was any difference between the treated and control groups in test case density in terms of the ratio of the defined test cases per requirements.
<b>Scope</b>	Small software development projects
<b>Attribute to Measure</b>	Test Coverage
<b>Metric Scale</b>	Interval
<b>Related Measure(s)</b>	Number of Defined Test Cases (Measure 12), Number of Requirements (Measure 2)
<i>Required Measurement</i>	
No. of Defined Test Cases (NDTC)	$\frac{NDTC}{NR}$
No. of Requirements (NR)	

**Metric 11** Test Case per Requirement Ratio

The following tables present the measures (direct metrics) used to evaluate the values of the main metrics stated above.

<b>Number of Traceable Requirements</b>	
<b>Definition</b>	It is a measure of the number of the requirements that are traceable.
<b>Type</b>	Qualitative
<b>Source</b>	Experiment Subjects
<b>Applicable Phase</b>	Requirement Analysis
<b>Purpose/Objective</b>	This measure is used in the experiment to determine the percentage of traceable requirement.
<b>Measurement Scale</b>	Interval
<b>Measurement Method</b>	Requirements in the requirement specification are individually read and checked for traceability. A requirement is traceable if it can be linked to its source and the related design, test, and implementation [Davis 1993]. Total number of traceable requirements are counted and recorded.
<b>Related Metrics</b>	Percentage of Traceable Requirements (Metric 1)

**Measure 1** Number of Traceable Requirements

<b>Number of Requirements</b>	
<b>Definition</b>	It is a measure of the number of defined requirements.
<b>Type</b>	Quantitative
<b>Source</b>	Experiment Subjects
<b>Applicable Phase</b>	Requirement Analysis
<b>Purpose/Objective</b>	This measure is used in the experiment to determine the value of the two metrics - Percentage of Traceable Requirements, and Test Case Density (Test Case per Requirement Ratio)
<b>Measurement Scale</b>	Interval
<b>Related Metrics</b>	Percentage of Traceable Requirements (Metric 1), Test Case per Requirement Ratio (Metric 11)

**Measure 2** Number of Requirements

<b>Number of Defects Detected</b>	
<b>Definition</b>	It is a measure of the number of defects detected in a development phase.
<b>Type</b>	Quantitative
<b>Source</b>	Experiment Subjects
<b>Applicable Phase</b>	Requirement analysis, Design, Implementation, and Delivery
<b>Objective</b>	This measure, in conjunction with 'number of defects fixed' measure, is used to work out the 'percentage of defects fixed' as an indication of defect correction quality.
<b>Measurement Scale</b>	Interval
<b>Notes</b>	Defect detection is done by the developers (i.e. experiment subject). Number of defects detected is dependent on the thoroughness with which the reviewer carries out the reviews (i.e. the more thorough the reviewer, the more likely to detect any defects). However due to the random selection of the subjects into experimental and control groups and the relatively high number of subjects, the thoroughness of defect detection process is taken to be a constant across the treated and control groups and therefore does not effect the objective of this measure.
<b>Related Metrics</b>	Defect Density (Metric 9), Percentage of Defects Fixed (Metric 2)

## Measure 3 Number of Detected Defects

<b>Number of Defects Fixed</b>	
<b>Definition</b>	It is a measure of the number of defects fixed in a development phase.
<b>Type</b>	Quantitative
<b>Source</b>	Experiment Subjects
<b>Applicable Phase</b>	Requirement analysis, Design, Implementation, and Delivery
<b>Purpose/Objective</b>	This measure is used to calculate 'the percentage of defects fixed'.
<b>Measurement Scale</b>	Interval
<b>Related Metrics</b>	Percentage of Defects Fixed (Metric 2)

## Measure 4 Number of Defects Fixed

<b>Phase Time</b>	
<b>Definition</b>	It is a measure of the time (person-hour) spent in a development phase
<b>Type</b>	Quantitative
<b>Source</b>	The value for this measure is provided by the experiment subjects
<b>Applicable Phase</b>	Requirement analysis, Design, Implementation, and Delivery
<b>Measurement Scale</b>	Interval
<b>Purpose/Objective</b>	This measure is used in the experiment to calculate percentage phase time spent in the development phase
<b>Related Metrics</b>	Percentage of Phase Time Spent on Testing (Metric 4)

## Measure 5 Time Spent in a Development Phase

<b>Development Time</b>	
<b>Definition</b>	It is a measure of the time (person-hour) spent in the development project. It is the sum of time spent in RA, Design, Implementation, and Delivery phases.
<b>Type</b>	Quantitative
<b>Source</b>	The value for this measure is provided by the experiment subjects
<b>Measurement Scale</b>	Interval
<b>Purpose/Objective</b>	This measure is used in the experiment to calculate overall productivity.
<b>Related Metrics</b>	Productivity (Metric 7)

**Measure 6 Total Time Spent on Development Project**

<b>Phase Test Time</b>	
<b>Definition</b>	It is a measure of the time (person-hour) spent on testing in a development phase.
<b>Type</b>	Quantitative
<b>Source</b>	Experiment Subjects
<b>Applicable Phase</b>	Requirement analysis, Design, Implementation, and Delivery
<b>Measurement Scale</b>	Interval
<b>Purpose/Objective</b>	This measure is used in the experiment to calculate the percentage of phase time spent in tests
<b>Related Metrics</b>	Percentage of Phase Time Spent on Testing (Metric 4)

**Measure 7 Time Spent in Testing in a Development Phase**

<b>Size of Source Code (LOC)</b>	
<b>Definition</b>	It is a measure of the number of source lines of code. There are many different definitions and interpretation of LOC, and therefore many ways to count LOC. For the purpose of the experiment, a line of code is defined as follows:  "A line of code is any line of program text that is not a comment or blank line, regardless of the number of statements or fragments of statements on the line. This specifically includes all lines containing program headers, declarations, and executable and non-executable statements." [Conte 1986].
<b>Type</b>	Quantitative
<b>Source</b>	Experiment Subjects
<b>Applicable Phase</b>	Implementation
<b>Measurement Scale</b>	Interval
<b>Purpose/Objective</b>	This measure was used to work out to productivity and comment density.
<b>Notes</b>	LOC is one the oldest and most widely used software size measure [Sommerville 2007]. It has the advantage of being easy to collect - no other measure is as well-understood [Bassman et al. 1995]. It however suffers from some weaknesses such as, being language dependent, equating length as a measure of size without regards to complexity or functionality, discarding the fact that bad software designs may cause excessive lines of code (see Section 4.10). However, as the LOC weaknesses are randomly and universally spread between the control and treated groups in the experiment, they will not affect the objective of the experiment, which is to compare the treated and control groups.
<b>Related Metrics</b>	Comment density (Metric 10), Productivity (Metric 7)

**Measure 8 Size of Source Code (LOC)**

<b>Number of Classes</b>	
<b>Definition</b>	It is a measure of the number of classes developed
<b>Type</b>	Quantitative
<b>Source</b>	Experiment Subjects
<b>Applicable Phase</b>	Design
<b>Measurement Scale</b>	Interval
<b>Purpose/Objective</b>	This measure is used to determine methods per class ratio
<b>Related Metrics</b>	Methods per Class Ratio (Metric 6)

**Measure 9** Number of Classes

<b>Number of Methods</b>	
<b>Definition</b>	It is a measure of the number of methods developed
<b>Type</b>	Quantitative
<b>Source</b>	Experiment Subjects
<b>Applicable Phase</b>	Design
<b>Measurement Scale</b>	Interval
<b>Purpose/Objective</b>	This measure is used to determine methods per class ratio.
<b>Related Metrics</b>	Methods per Class Ratio (Metric 6)

**Measure 10** Number of Methods

<b>Number of Lines of Comment</b>	
<b>Definition</b>	It is a measure of the number of lines of comments.
<b>Type</b>	Quantitative
<b>Source</b>	Experiment Subjects
<b>Applicable Phase</b>	Implementation
<b>Measurement Scale</b>	Interval
<b>Purpose/Objective</b>	This measure is used to determine comment density.
<b>Related Metrics</b>	Comment density (Metric 10)

**Measure 11** Number of Lines of Comment

<b>Number of Defined Test Cases</b>	
<b>Definition</b>	It is a measure of the number of defined test cases
<b>Type</b>	Quantitative
<b>Source</b>	Experiment subjects
<b>Applicable Phase</b>	Delivery
<b>Measurement Scale</b>	Interval
<b>Purpose/Objective</b>	This measure is used in the experiment to calculate test effectiveness ratio
<b>Related Metrics</b>	Test case coverage (Metric 11)

**Measure 12** Number of Defined Test Cases



## Appendix D. Results

Statistical analysis results, when the individual and group projects are combined, are depicted in the following tables. The first set of tables presents the analysis of the metrics followed by analysis of official marks awarded to the four attributes of the software project (i.e. *product, design and analysis, evaluation, and project management*).

Operation	Independent Variable	Dependent Variable	Projects
Independent Samples t-test	Experiment group-type (Treated, and Control)	Software Attribute	Individual and Group Projects
Phase	Metric	P_ value	Comment
Requirement Analysis	Percentage of Traceable requirement	0.003	Value for treated groups was significantly higher than control groups. Therefore, positive effect of process patters is confirmed.
	Percentage of Reviewed Requirements Specification	0.019	Value for treated groups was significantly higher than control groups. Therefore, positive effect of process patters is confirmed.
	Percentage of Defects Fixed (RA Phase)	0.162	There was not a statistically significant difference between the treated and control groups.
	Percentage of phase time spent in testing	0.194	There was not a statistically significant difference between the treated and control groups.
Design	Percentage of design document reviewed	0.000	Value for control groups was significantly higher than treated groups. Therefore, positive effect of process patters is confirmed.
	Number of methods per class (Methods per Class Ratio)	0.001	Value for control groups was significantly higher than treated groups. Therefore, positive effect of process patters is confirmed.
	Percentage of Defects Fixed (Design Phase)	0.242	There was not a statistically significant difference between the treated and control groups.
	Percentage of phase time spent in testing (Design Phase)	0.097	There was not a statistically significant difference between the treated and control groups.
Implementation	Comment Density	0.025	Value for treated groups was significantly higher than control groups. Therefore, positive effect of process patters is confirmed.
	Percentage of Code Reviewed	0.011	Value for treated groups was significantly higher than control groups. Therefore, positive effect of process patters is confirmed.
	Productivity (Implementation phase)	0.001	Value for treated groups was significantly higher than control groups. Therefore, positive effect of process patters is confirmed.
	Productivity (complete development project)	0.003	Value for treated groups was significantly higher than control groups. Therefore, positive effect of process patters is confirmed.
	Percentage of defects fixed	0.197	There was not a statistically significant difference between the treated and control groups.
	Defect Density	0.012	Value for control groups was significantly higher than treated groups. Therefore, positive effect of process patters is confirmed.
	Percentage of implementation phase time spent in testing	0.000	Value for control groups was significantly higher than treated groups. Therefore, positive effect of process patters is confirmed.
	Test case density (Test case per Requirement)	0.001	Value for treated groups was significantly higher than control groups. Therefore, positive effect of process patters is confirmed.

Delivery	Percentage of defects fixed	0.003	Value for treated groups was significantly higher than control groups. Therefore, positive effect of process patters is confirmed.
	Percentage of Delivery phase time spent in testing	0.000	Value for treated groups was significantly higher than control groups. Therefore, positive effect of process patters is confirmed.

Table App\_D 1 Significance analysis results for metrics for individual and group projects combined

Operation	Independent Variable	Dependent Variable	Projects
Independent Samples t-test	Experiment group-type	Software Attribute	Individual and Group Projects
Software Attribute	P_ value	Comment	
Design and Analysis	0.182	There was not a statistically significant difference between the treated and control groups.	
Product	0.004	Value for treated groups was significantly higher than control groups. Therefore, positive effect of process patters is confirmed.	
Project Management	0.143	There was not a statistically significant difference between the treated and control groups.	
Evaluation	0.106	There was not a statistically significant difference between the treated and control groups.	

Table App\_D 2 Significance analysis results for tutor marks for individual and group projects combined

### Results of the conducted survey on software patterns

Question	Yes %	No %
Do you believe there are risks involved in using patterns	19	81
Do you use patterns in software development?	60	40
Do you write patterns	6	94
Do you publish patterns	6	94
Do you develop domain-specific patterns	3	97

Question (Pattern Users Only)	Never %	Seldom %	Frequently %	Always %
Do you validate patterns that you use	90	7	3	0

Question (Pattern Users Only)	External Evaluations %	Rationale %	Using test cases %	Other %
How do you validate patterns that you use	0	0	10	0

What types of pattern do you use and in what capacity (Pattern Users Only)				
Pattern Type	Never %	Seldom %	Frequently %	Extensively %
Analysis Patterns	97	0	3	0
Design Patterns	0	15	63	22
Process patterns	85	12	3	0

Where do you get your patterns (Pattern Users Only)				
Pattern Type	Never %	Seldom %	Frequently %	Extensively %
In-house produced	90	0	10	0
Books	13	17	42	28
Journals, Conf. Proceedings	45	40	12	3
Pattern Community (repositories)	30	35	28	7

Do you have concerns about using patterns because (Pattern Users Only)				
Pattern Type	None %	Slightly %	Moderately %	Extremely %
Patterns could be outdated	22	60	18	0
Patterns could have unknown side effects	18	47	35	0
Your team may not be sufficiently proficient in patterns	20	45	32	3

**Results of *patterncentral.com* website survey**  
**Curtsy of *Patternscentral.com***

**Are patterns just hype or do they provide great value? (Total Votes: 572)**

Yes, big time hype	8.2%
Patterns are definitely valuable	49.8%
Patterns are valuable, but they tend to be misused	31.8%
Don't know, but I want to learn more	8.4%
I couldn't really care less	1.8%

**Does your organisation support the use of patterns? (Total Votes: 375)**

Yes, and we know what we're doing	29.6%
Yes, but patterns aren't really understood	31.5%
No	10.4%
I work alone and use patterns where appropriate	25.6%
I work alone and don't use patterns	2.9%

**Are you actively using Patterns in your software development? (Total Votes: 988)**

What's a Pattern?	5.6%
Have read about them	12.0%
Sometimes	23.2%
Whenever possible	59.2%

### Some of the views expressed in the survey (Chapter 3) by architects on architectural patterns

"My own view is that the book was a simplistic attempt to link behaviour to form, and had a underlying 'romantic' agenda which prioritised a particular, traditional vocabulary (Arts and Crafts especially) over less 'aesthetic' forms of architecture (Brutalism for example). In short, a work of its time"

[Dr. Vaughan Hart Bath University]

"... While sharing some of their criticisms, I wouldn't be so dismissive as my colleagues about the book. Some of the ideas in it have percolated quite far into diffuse thinking about buildings, getting into the heads of many a solo architect or small practices, and leading them to be more observant, even if they may not be aware of the source." [Dr. Mark Wilson Jones] Bath University

"... I do feel that this is timeless, but not so fashionable. It will come around again as people re-discover the social/humanist agenda" [Fiona McLachlan, Head of architecture, University of Edinburgh]

"... The book I think is considered rather old-fashioned - even naive."

[Prof. Robert Kronenburg, Head of school of architecture, University of Liverpool]

"... whilst Alexander's language is extremely useful to describe buildings from a technological or even functional standpoint, it is not particularly well suited for the conceptualisation of buildings from an experiential point of view." [Carlos Calderon, Glamorgan University]

"I certainly use 'A Pattern Language' as a text for my Theory of Landscape Architecture course, but I am always anxious to stress that it should be used as an 'ideas book', a stimulus to creative thinking, rather than some kind of infallible recipe book" [Dr. Ian H. Thompson, University of Newcastle]

"My opinion of the book is that it is outdated, and, even when published, prescriptive in the wrong way. Architectural design and theory has moved a very long way since the book was published"

[Prof. Mike Jenks, Head of Department, Oxford Brookes University]

"I think that it is now considered rather old and tired. Certainly never 'taught' in a school of architecture in my experience". [Todd Wakefield, Head School of Architecture. University of Portsmouth]

"My own view of the pattern language is that it is too prescriptive and relies too heavily on normalised views of human behaviour. If applied rigorously it removes many opportunities for creativity in design" [Dr. Christopher Tweed, Queen's University Belfast]

".....in the 'pattern language', what is and is not a pattern seems to be decided by Alexander"

[Professor Bill Hillier, Space Syntax UCL]

"I co-ordinate first year studio, and steer well clear of Pattern Language"

[Stephen Walker Sheffield University]

"1. Alexander not well liked, not "designery"..... not enough aesthetics, too much about feelings  
2. Alexander (like me) is no relativist - what is "opinion" to most critics is objective fact to others - I mean stuff like that we all yearn for sunlight, enclosure, rhythm, boundary etc  
3. Pattern Language gives you a structure to understand a human-based approach to design - as opposed to the standard one led by expediency, egotism, detail etc"

[Malcolm Fraser, Leading Architect, Edinburgh University]

"A Pattern Language offers descriptions of all the parts, but not a description or understanding of how those parts can come together to produce social meaning; while it is beautifully written and illustrated, it does not teach how to design"

[Tim Stonor, Space Syntax UCL]

## Appendix E. Survey Questionnaires

---

The questionnaires for the two surveys are presented in this appendix.

- Survey of UK software development companies on their usage of patterns
- Survey of UK universities on their views and teachings of architectural patterns

## Survey Questionnaire on Software Patterns

### Section A. General

**A1) How many employees does your organisation have?**

Less than 10  10 – 50  51 – 100  101-200  200+

**A2) What Capability Maturity Model (CMM/SPICE) Level is your development process is currently at?**

Not Measured  Level 1  Level 2  Level 3  Level 4  Level 5

**A3) Is your company ISO9000 (or other ISO standards) registered?**

Yes  No  Others

**A4) What programming languages do you use for software development?**

None  Java  C++  C#  VB  Others

If others please state :

**A5) What scripting languages do you use for software development?**

None  JSP  ASP  PHP  HTML  Others

If others please state

**A6) What do you believe to be the effect of application of patterns on the following software quality attributes?**

Reliability	Positive <input type="checkbox"/>	Negative <input type="checkbox"/>	Neutral <input type="checkbox"/>	Don't Know <input type="checkbox"/>
Usability	Positive <input type="checkbox"/>	Negative <input type="checkbox"/>	Neutral <input type="checkbox"/>	Don't Know <input type="checkbox"/>
Changeability	Positive <input type="checkbox"/>	Negative <input type="checkbox"/>	Neutral <input type="checkbox"/>	Don't Know <input type="checkbox"/>
Interoperability	Positive <input type="checkbox"/>	Negative <input type="checkbox"/>	Neutral <input type="checkbox"/>	Don't Know <input type="checkbox"/>
Efficiency	Positive <input type="checkbox"/>	Negative <input type="checkbox"/>	Neutral <input type="checkbox"/>	Don't Know <input type="checkbox"/>
Reusability	Positive <input type="checkbox"/>	Negative <input type="checkbox"/>	Neutral <input type="checkbox"/>	Don't Know <input type="checkbox"/>
Testability	Positive <input type="checkbox"/>	Negative <input type="checkbox"/>	Neutral <input type="checkbox"/>	Don't Know <input type="checkbox"/>
Portability	Positive <input type="checkbox"/>	Negative <input type="checkbox"/>	Neutral <input type="checkbox"/>	Don't Know <input type="checkbox"/>
Maintainability	Positive <input type="checkbox"/>	Negative <input type="checkbox"/>	Neutral <input type="checkbox"/>	Don't Know <input type="checkbox"/>

**A7) Do you believe patterns contribute towards better communication between software development team members**

Yes  No  Don't Know

**A8) Do you believe there are risks involved in using patterns**

Nil  Slight  Moderate  Considerable  Don't Know

**A9) Does your firm use patterns in software development?**

Yes  No

*If No Go to Section D, Future Plan*

**A10) Does your firm generate (produce, write) patterns?**

Yes  No

*If No go to Section C, Pattern Usage*

**Section B. Pattern Development**

Please fill in this section if your organisation generate (write) patterns

**B1) Does your firm publish externally the patterns it develops?**

Yes  No

**B2) Does your firm develop domain-specific patterns (e.g. Telecommunication)?**

No  Yes  Name [ \_\_\_\_\_ ]

**B3) Do you have a repository of in-house developed patterns?**

Analysis Patterns	0 <input type="checkbox"/>	1 - 10 <input type="checkbox"/>	11 - 20 <input type="checkbox"/>	21 - 50 <input type="checkbox"/>	50+ <input type="checkbox"/>
Design Patterns	0 <input type="checkbox"/>	1 - 10 <input type="checkbox"/>	11 - 20 <input type="checkbox"/>	21 - 50 <input type="checkbox"/>	50+ <input type="checkbox"/>
Process patterns	0 <input type="checkbox"/>	1 - 10 <input type="checkbox"/>	11 - 20 <input type="checkbox"/>	21 - 50 <input type="checkbox"/>	50+ <input type="checkbox"/>
Other patterns	0 <input type="checkbox"/>	1 - 10 <input type="checkbox"/>	11 - 20 <input type="checkbox"/>	21 - 50 <input type="checkbox"/>	50+ <input type="checkbox"/>

Please State:

**Section C. Pattern Usage**

Please fill in this section if your organisation uses patterns.

**C1) What types of pattern do you use and in what capacity?**

Analysis Patterns	Never <input type="checkbox"/>	Seldom <input type="checkbox"/>	Frequently <input type="checkbox"/>	Extensively <input type="checkbox"/>
Design Patterns	Never <input type="checkbox"/>	Seldom <input type="checkbox"/>	Frequently <input type="checkbox"/>	Extensively <input type="checkbox"/>
Process Patterns	Never <input type="checkbox"/>	Seldom <input type="checkbox"/>	Frequently <input type="checkbox"/>	Extensively <input type="checkbox"/>
Others	Never <input type="checkbox"/>	Seldom <input type="checkbox"/>	Frequently <input type="checkbox"/>	Extensively <input type="checkbox"/>

If Others please state where:

**C2) Where do you get your patterns?**

In-house produced	Never <input type="checkbox"/>	Seldom <input type="checkbox"/>	Frequently <input type="checkbox"/>	Extensively <input type="checkbox"/>
Books	Never <input type="checkbox"/>	Seldom <input type="checkbox"/>	Frequently <input type="checkbox"/>	Extensively <input type="checkbox"/>
Journals	Never <input type="checkbox"/>	Seldom <input type="checkbox"/>	Frequently <input type="checkbox"/>	Extensively <input type="checkbox"/>
Pattern Community	Never <input type="checkbox"/>	Seldom <input type="checkbox"/>	Frequently <input type="checkbox"/>	Extensively <input type="checkbox"/>
Business Partners	Never <input type="checkbox"/>	Seldom <input type="checkbox"/>	Frequently <input type="checkbox"/>	Extensively <input type="checkbox"/>

Others                      Never     Seldom     Frequently     Extensively

If Others please state where:

**C3) If you use any of the patterns below, how do you rate their ease-of-use?**

Analysis Patterns    Easy     Moderate     Difficult     Very difficult   
 Design Pattern        Easy     Moderate     Difficult     Very difficult   
 Process Patterns      Easy     Moderate     Difficult     Very difficult

**C4) If you use any of the patterns below, how do you rate their usefulness?**

Analysis Patterns    Nil     Slight     Moderate     Considerable   
 Design Patterns      Nil     Slight     Moderate     Considerable   
 Process Patterns     Nil     Slight     Moderate     Considerable

**C5) Do you have concerns about using patterns because:**

Patterns could be outdated.  
                                   No Concerns |  |  |  |  | Extreme Concerns  
 Patterns could have unknown side effects.  
                                   No Concerns |  |  |  |  | Extreme Concerns  
 Your team may not be sufficiently proficient in patterns.  
                                   No Concerns |  |  |  |  | Extreme Concerns  
 Others  
                                   No Concerns |  |  |  |  | Extreme Concerns

If Others please state the reason:

**C6) Do you validate patterns that you use by testing or other methods?**

Never             Seldom             Frequently             Always

**C7) How do you validate patterns that you use**

External Evaluations   
 Rationale   
 Using test cases   
 Other (Please specify)



**Section D. Future Plan**

**D1) You do not use patterns in your company because:**

You believe patterns do not provide an advantage (technical or economic)	Yes <input type="checkbox"/>	No <input type="checkbox"/>
You do not have the skill set	Yes <input type="checkbox"/>	No <input type="checkbox"/>
Patterns may be outdated	Yes <input type="checkbox"/>	No <input type="checkbox"/>
You do not trust patterns to provide the best solution	Yes <input type="checkbox"/>	No <input type="checkbox"/>
Your software development practices do not require patterns	Yes <input type="checkbox"/>	No <input type="checkbox"/>
You believe patterns could have adverse side effects	Yes <input type="checkbox"/>	No <input type="checkbox"/>

Other Reasons (Please Specify)

**D2) Does your firm plan to use patterns in the future?**

No plans  Next 3 months  Next 6 Months  Next 12 Months

**Section E. Comments**

**Please state below any comments that you would like to make on patterns.**

**Questionnaire to the Architecture Departments of UK universities**

Dear Sir,

As part of a PhD level research at Middlesex University, we are evaluating the works of Christopher Alexander on patterns - specifically his book "A Pattern Language". I would therefore be grateful if you would kindly answer the following two questions:

Q. 1) Do you teach pattern languages, as described in the book 'A Pattern Language' by Christopher Alexander, in your department, in any undergraduate or postgraduate courses and at what level of usage?

None	<input type="checkbox"/>	Undergraduate	<input type="checkbox"/>	Postgraduate	<input type="checkbox"/>
		Low	<input type="checkbox"/>	Low	<input type="checkbox"/>
		Moderate	<input type="checkbox"/>	Moderate	<input type="checkbox"/>
		High	<input type="checkbox"/>	High	<input type="checkbox"/>

Q. 2) What are your views on the philosophy and concept of Alexander's pattern languages?

No views     Negative     Neutral     Positive

**Please Comment.**

Thanks very much for your help.

Best Regards

Ahmad Estabraghy  
 Computer Science Dept,  
 Middlesex University,  
 London.