Copyright is owned by the Author of the thesis. Permission is given for a copy to be downloaded by an individual for the purpose of research and private study only. The thesis may not be reproduced elsewhere without the permission of the Author.
M.O.O.T

META OBJECT
ORIENTATED TOOL

A NOVEL META-CASE TOOL METHODOLOGY
REPRESENTATION STRATEGY

A dissertation submitted in partial fulfilment of the requirements for the degree of Doctor of Philosophy in Computer Science

Massey University, New Zealand

David Charles Page

1998
Abstract

This thesis presents an investigation into current meta-CASE technology. The research focuses on CASE tool support for the concept of methodology, the representation of methodology syntax and semantics, and the support for re-use of methodology descriptions and software artefacts. A novel methodology representation strategy for meta-CASE tools is proposed and implemented with the development of a new meta-CASE tool (MOOT – Meta Object Orientated Tool).

The novel strategy propounded in this thesis uses an object-orientated meta-model and views methodology descriptions as potentially re-usable components. The coupling between methodology syntax and semantic descriptions is minimised so they can be re-used independently.

Two new modelling languages have been derived, to support the definition of syntax (NDL – Notation Definition Language) and semantics (SSL – Semantic Specification Language) of software engineering methodologies. Semantic descriptions are compiled to a platform independent representation (SSL-BC), which is executed on a purpose built virtual machine (SSL-VM). Late binding of syntax and semantic methodology descriptions is implemented with the development of Notation Semantic Mapping (NSM) tables. Two libraries of re-usable methodology description components, the Core Knowledge Base (CKB) and the Generic Object Orientated Knowledge Base (GOOKB), have been derived during this research.

Empirical results gained from applying the MOOT prototype demonstrated the flexibility, extensibility and potential of the novel methodology representation strategy. This approach permitted the implementation and modelling of UML and patterns, two recent advances of object technology that did not exist when the research commenced.

The novel strategy presented in this thesis is more than an untried theory. It has been implemented, applied and is being evaluated. Simply, it is real and it works.
DEDICATION

This thesis is lovingly dedicated to my parents

Michael Julius Page and Susan Evelyn Page
# TABLE OF CONTENTS

## INTRODUCTION

1.1 Introduction  
1.2 Fundamental Terms  
1.2.1 Software Engineering Development Methodology  
1.2.2 Meta-Modelling  
1.2.3 Computer Aided Software Engineering (CASE)  
1.2.4 CASE Tool  
1.2.5 Meta-CASE and Meta-CASE Tool  
1.3 Object-Orientated Software Development Methodologies  
1.4 CASE Technology  
1.5 Methodology CASE Tools  
1.5.1 Methodology Dependent CASE Tools  
1.5.2 Multi-Methodology CASE Tools  
1.5.3 Tools that Support More than One Methodology  
1.5.4 Meta-CASE Tools  
1.5.5 CASE Tool Generators  
1.5.6 Modifiable CASE Environments  
1.6 Limitations of Methodology CASE Tools  
1.6.1 Limitations from the Organisational Perspective  
1.6.2 Limitations from the CASE Tool Perspective  
1.7 Objectives of the Research  
1.8 Method  
1.9 Outline of the Thesis  

## META-MODELLING AND META-CASE TOOLS

2.1 Introduction  
2.2 Meta-Modelling  
2.2.1 The OMG Meta Object Facility  
2.2.2 Unified Modelling Language  
2.2.3 COMMA  
2.2.4 Open Modelling Language  
2.2.5 OOram  
2.2.6 CASE Data Interchange Format  
2.2.7 ISO/CDIF Meta-Model  

4.4.6 Actions

4.5 Notation Definition Language
  4.5.1 Requirements of NDL
  4.5.2 Design of NDL
  4.5.3 Describing Symbols in NDL
  4.5.4 Support for Grouping
  4.5.5 Docking Areas
  4.5.6 Describing Connections in NDL

4.6 NDL Interpreter

4.7 Design of the NDL Interpreter
  4.7.1 Representing Expressions
  4.7.2 Segment Templates
  4.7.3 Group Templates
  4.7.4 Connection and Symbol Templates

4.8 Implementation of the NDL Interpreter

4.9 Summary

SEMANTIC SPECIFICATION LANGUAGE

5.1 Introduction

5.2 Method

5.3 Rationale and Goals of SSL

5.4 Requirements of SSL

5.5 Semantic Specification Language
  5.5.1 Overview
  5.5.2 MOOT Meta-Model
  5.5.3 Module System
  5.5.4 Memory Management
  5.5.5 Messages

5.6 Semantic Specification Language Definition
  5.6.1 Collections
  5.6.2 Simple Expressions
  5.6.3 Interface Module
  5.6.4 Class Interface Definition
  5.6.5 Implementation Module
  5.6.6 Class Definition
  5.6.7 Methods
  5.6.8 Statements

5.7 SSL Compiler

5.8 Executing SSL
5.9 SSL Virtual Machine 145
   5.9.1 Requirements of the SSL Virtual Machine 146
   5.9.2 Architecture of the SSL Virtual Machine 146
   5.9.3 SSL Virtual Machine Instruction Set 147
   5.9.4 Internal Representation of Classes, Objects and Methods 148
   5.9.5 Processing Messages on the Virtual Machine 151

5.10 Summary 153

THE CORE KNOWLEDGE BASE AND GENERIC OBJECT ORIENTATED KNOWLEDGE BASE 154

6.1 Introduction 154

6.2 Context of the Core Knowledge Base and the Generic Object Orientated Knowledge Base 154

6.3 Development of the Core Knowledge Base 156
   6.3.1 Meta-Model of Methodology 156
   6.3.2 Meta-Model of Modelling Language 158
   6.3.3 Handling Exceptional Situations 165

6.4 Development of the Generic Object Orientated Knowledge Base 166
   6.4.1 Object-Orientated Methodology Comparisons 167
   6.4.2 Method used to Design the Generic Object Orientated Knowledge Base 170
   6.4.3 Generic Object Orientated Knowledge Base 171

6.5 Implementing the Knowledge Bases 175

6.6 Summary 176

REALISING METHODOLOGIES AND SOFTWARE ENGINEERING PROJECTS IN MOOT 177

7.1 Introduction 177

7.2 Interaction Between CASE Tool Clients and the MOOT Core 177
   7.2.1 CASE Tool Client Requests 178
   7.2.2 MOOT Core Directives and Responses 180

7.3 Methodology Description Table 181
   7.3.1 Composition of the Methodology Description Table 181
   7.3.2 Applying the Methodology Description Table 184

7.4 Notation Semantic Mapping Tables 185
   7.4.1 NDL vs. SSL 185
   7.4.2 Composition of NSM Tables 187
7.4.3 Applying NSM Tables

7.5 Summary

VALIDATING THE MOOT APPROACH

8.1 Introduction

8.2 Defining the Coad and Yourdon Methodology

8.3 Supporting Patterns

8.4 Supporting UML

8.5 Preliminary Development of the Semantics Editor

8.5.1 Notation for the SSL Module Structure Modelling Language

8.5.2 Notation for the SSL Method Modelling Language

8.6 Toward Supporting Joosten Workflow Modelling

8.7 Summary

CONCLUSION AND FUTURE WORK

9.1 Introduction

9.2 Summary of the Thesis

9.3 Discussion

9.3.1 The Novel Meta-CASE Tool Methodology Representation Strategy

9.3.2 The MOOT Approach

9.3.3 The Notation Definition Language

9.3.4 The Semantic Specification Language

9.3.5 Core Knowledge Base and Generic Object Orientated Knowledge Base

9.4 Future Work

9.4.1 The Notation Definition Language

9.4.2 The Semantic Specification Language

9.4.3 Notation Semantic Mapping Tables

9.4.4 Support for Re-use

9.4.5 Cognitive Support

9.4.6 Meta-Modelling

9.4.7 Validation of a Complete Implementation of MOOT

9.5 Conclusion
# APPENDICES

## EVALUATION FRAMEWORK

<table>
<thead>
<tr>
<th>Section</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>I.1</td>
<td>Existing Evaluation Framework</td>
<td>246</td>
</tr>
<tr>
<td>I.2</td>
<td>A New Evaluation Framework</td>
<td>246</td>
</tr>
</tbody>
</table>

## NDL GRAMMAR

<table>
<thead>
<tr>
<th>Section</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>II.1</td>
<td>Introduction</td>
<td>249</td>
</tr>
<tr>
<td>II.2</td>
<td>Reserved Words</td>
<td>249</td>
</tr>
<tr>
<td>II.3</td>
<td>Operators</td>
<td>249</td>
</tr>
<tr>
<td>II.4</td>
<td>Grammar</td>
<td>249</td>
</tr>
</tbody>
</table>

## SSL GRAMMAR

<table>
<thead>
<tr>
<th>Section</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>III.1</td>
<td>Introduction</td>
<td>253</td>
</tr>
<tr>
<td>III.2</td>
<td>Reserved Words</td>
<td>253</td>
</tr>
<tr>
<td>III.3</td>
<td>Operators</td>
<td>253</td>
</tr>
<tr>
<td>III.4</td>
<td>Grammar</td>
<td>253</td>
</tr>
</tbody>
</table>

## SSL EXAMPLES

<table>
<thead>
<tr>
<th>Section</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>IV.1</td>
<td>The Sieve of Eratosthenes Version 1</td>
<td>258</td>
</tr>
<tr>
<td>IV.1.1</td>
<td>Interface Module</td>
<td>259</td>
</tr>
<tr>
<td>IV.1.2</td>
<td>Implementation Module</td>
<td>259</td>
</tr>
<tr>
<td>IV.2</td>
<td>The Sieve of Eratosthenes Version 2</td>
<td>265</td>
</tr>
<tr>
<td>IV.2.1</td>
<td>Interface Module</td>
<td>265</td>
</tr>
<tr>
<td>IV.2.2</td>
<td>Implementation Module</td>
<td>266</td>
</tr>
</tbody>
</table>

## SSL-VM INSTRUCTION SET

<table>
<thead>
<tr>
<th>Section</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>V.1</td>
<td>Introduction</td>
<td>269</td>
</tr>
<tr>
<td>V.2</td>
<td>Instruction Set</td>
<td>269</td>
</tr>
</tbody>
</table>
SSL COMPILER

VI.1 Introduction 277
VI.2 The SSL Compiler 277
VI.3 Representing Types in the SSL Compiler 279
VI.4 Representing Statements and Expressions in the SSL Compiler 280
VI.5 Representing Modules in the SSL Compiler 282

THE SSL VIRTUAL MACHINE

VII.1 Introduction 285
VII.2 The SSL Virtual Machine 285
VII.3 Representing SSL Types 287
VII.4 SSL Proxies 287
VII.5 Processing Messages 289
VII.6 Binding 291
VII.7 Garbage Collection 293

REFERENCES 297
<table>
<thead>
<tr>
<th>Number</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Figure 1-1</td>
<td>Modelling</td>
<td>4</td>
</tr>
<tr>
<td>Figure 1-2</td>
<td>Meta-modelling</td>
<td>6</td>
</tr>
<tr>
<td>Figure 1-3</td>
<td>Classification hierarchy of CASE tool categories</td>
<td>14</td>
</tr>
<tr>
<td>Figure 1-4</td>
<td>Thesis outline</td>
<td>26</td>
</tr>
<tr>
<td>Figure 2-1</td>
<td>Four layer meta-modelling process</td>
<td>28</td>
</tr>
<tr>
<td>Figure 2-2</td>
<td>CDIF Meta-metamodel (EIA CDIF, 1994b)</td>
<td>35</td>
</tr>
<tr>
<td>Figure 2-3</td>
<td>CASE tool generators</td>
<td>37</td>
</tr>
<tr>
<td>Figure 2-4</td>
<td>Architecture of a modifiable CASE environment</td>
<td>38</td>
</tr>
<tr>
<td>Figure 2-5</td>
<td>Meta-CASE tools and the four layer meta-modelling architecture</td>
<td>39</td>
</tr>
<tr>
<td>Figure 3-1</td>
<td>Mapping between goals and design decisions made regarding MOOT</td>
<td>60</td>
</tr>
<tr>
<td>Figure 3-2</td>
<td>The relation between software projects, methodology descriptions and the description languages in MOOT</td>
<td>62</td>
</tr>
<tr>
<td>Figure 3-3</td>
<td>Methodology descriptions and software engineering projects</td>
<td>64</td>
</tr>
<tr>
<td>Figure 3-4</td>
<td>Knowledge bases in MOOT</td>
<td>65</td>
</tr>
<tr>
<td>Figure 3-5</td>
<td>The relation between the CKB, the GOOKB, methodologies and software engineering projects in MOOT</td>
<td>66</td>
</tr>
<tr>
<td>Figure 3-6</td>
<td>Meta-modelling architecture</td>
<td>67</td>
</tr>
<tr>
<td>Figure 3-7</td>
<td>Addressing the limitations of meta-CASE tools</td>
<td>69</td>
</tr>
<tr>
<td>Figure 3-8</td>
<td>The two roles of the MOOT system</td>
<td>71</td>
</tr>
<tr>
<td>Figure 3-9</td>
<td>Moot system</td>
<td>72</td>
</tr>
<tr>
<td>Figure 3-10</td>
<td>Proposed, top level, system architecture</td>
<td>73</td>
</tr>
<tr>
<td>Figure 3-11</td>
<td>Architecture of the MOOT prototype</td>
<td>79</td>
</tr>
<tr>
<td>Figure 4-1</td>
<td>A state transition diagram drawn in the notation of Booch and Feylock</td>
<td>84</td>
</tr>
<tr>
<td>Figure 4-2</td>
<td>A simple diagram drawn with UML, Coad and Yourdon and Booch notations</td>
<td>86</td>
</tr>
<tr>
<td>Figure 4-3</td>
<td>Three examples of a UML class symbol</td>
<td>87</td>
</tr>
<tr>
<td>Figure 4-4</td>
<td>Topographical description of a UML class</td>
<td>87</td>
</tr>
<tr>
<td>Figure 4-5</td>
<td>Coad and Yourdon and Booch symbols showing common sub-parts</td>
<td>88</td>
</tr>
<tr>
<td>Figure 4-6</td>
<td>Two example connections</td>
<td>88</td>
</tr>
<tr>
<td>Figure 4-7</td>
<td>Inheritance connection in UML</td>
<td>90</td>
</tr>
</tbody>
</table>
Figure 4-8 - An example UML sequence diagram
Figure 4-9 - A Jacobson Use Case diagram
Figure 4-10 - Docking areas on Coad and Yourdon Class&Object symbols
Figure 4-11 - Coad and Yourdon Subject Area: expanded (left) and collapsed (right)
Figure 4-12 - A UML class expressed with varying levels of detail
Figure 4-13 - Two example active areas
Figure 4-14 - (i) A symbol (ii) exploded Symbol (iii) templates
Figure 4-15 - Applying a template
Figure 4-16 - Topographical description of a UML class symbol
Figure 4-17 - A UML class symbol with active areas
Figure 4-18 - Template describing a UML class symbol
Figure 4-19 - Coad and Yourdon class and Class&Object symbols
Figure 4-20 - Identified common sub-parts in Coad and Yourdon's notation
Figure 4-21 - Group templates
Figure 4-22 - Coad and Yourdon class symbol
Figure 4-23 - Docking at a point
Figure 4-24 - Anatomy of a point docking area
Figure 4-25 - Docking on a line
Figure 4-26 - Anatomy of a line docking area
Figure 4-27 - Representing valid directions for a line docking area
Figure 4-28 - Docking on an arc
Figure 4-29 - Anatomy of an arc docking area
Figure 4-30 - Representing valid directions for an arc docking area
Figure 4-31 - Two example connections
Figure 4-32 - Connection symbol template
Figure 4-33 - Coad and Yourdon connection symbol line docking area (i) with a single connection (ii) with multiple connections
Figure 4-34 - Connection terminator templates for Coad and Yourdon Gen-Spec and message connections
Figure 4-35 - NDL connection templates for Coad and Yourdon Gen-Spec and message connections.
Figure 4-36 - Components of the NDL interpreter
Figure 4-37 - The Expression class hierarchy
Figure 4-38 - Template segment hierarchy
<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>4-39</td>
<td>The different types of template</td>
<td>118</td>
</tr>
<tr>
<td>4-40</td>
<td>NDL interpreter using an NDL description of the Rumbaugh instance and object diagram</td>
<td>119</td>
</tr>
<tr>
<td>4-41</td>
<td>NDL interpreter using an NDL description of the Coad and Yourdon class diagram</td>
<td>120</td>
</tr>
<tr>
<td>5-1</td>
<td>Mapping between goals and design decisions made regarding features of SSL</td>
<td>124</td>
</tr>
<tr>
<td>5-2</td>
<td>MOOT meta-metamodel</td>
<td>127</td>
</tr>
<tr>
<td>5-3</td>
<td>The built-in SSL variables</td>
<td>128</td>
</tr>
<tr>
<td>5-4</td>
<td>SSL collection and iterator types</td>
<td>131</td>
</tr>
<tr>
<td>5-5</td>
<td>Partial SSL implementation of a list class</td>
<td>135</td>
</tr>
<tr>
<td>5-6</td>
<td>SSL implementation of the list class</td>
<td>137</td>
</tr>
<tr>
<td>5-7</td>
<td>Implementing SSL create operations</td>
<td>138</td>
</tr>
<tr>
<td>5-8</td>
<td>Example loop and if statements</td>
<td>140</td>
</tr>
<tr>
<td>5-9</td>
<td>SSL compiler</td>
<td>141</td>
</tr>
<tr>
<td>5-10</td>
<td>Processing actions</td>
<td>143</td>
</tr>
<tr>
<td>5-11</td>
<td>Architecture of the SSL virtual machine</td>
<td>146</td>
</tr>
<tr>
<td>5-12</td>
<td>SSL class</td>
<td>149</td>
</tr>
<tr>
<td>5-13</td>
<td>SSL method</td>
<td>150</td>
</tr>
<tr>
<td>5-14</td>
<td>SSL object</td>
<td>151</td>
</tr>
<tr>
<td>5-15</td>
<td>Processing messages on the SSL-VM</td>
<td>152</td>
</tr>
<tr>
<td>6-1</td>
<td>The three tier structure of the information processed by MOOT</td>
<td>155</td>
</tr>
<tr>
<td>6-2</td>
<td>Methodology meta-model</td>
<td>157</td>
</tr>
<tr>
<td>6-3</td>
<td>Transitions</td>
<td>158</td>
</tr>
<tr>
<td>6-4</td>
<td>Meta-model of modelling language</td>
<td>159</td>
</tr>
<tr>
<td>6-5</td>
<td>Representing a whole-part relation</td>
<td>160</td>
</tr>
<tr>
<td>6-6</td>
<td>Representing a class diagram with instances of classes from the CKB</td>
<td>161</td>
</tr>
<tr>
<td>6-7</td>
<td>Detailed meta-model of modelling language</td>
<td>162</td>
</tr>
<tr>
<td>6-8</td>
<td>Extended meta-model of modelling language</td>
<td>163</td>
</tr>
<tr>
<td>6-9</td>
<td>Core Knowledge Base</td>
<td>164</td>
</tr>
<tr>
<td>6-10</td>
<td>Situations</td>
<td>165</td>
</tr>
<tr>
<td>6-11</td>
<td>Critics</td>
<td>166</td>
</tr>
<tr>
<td>6-12</td>
<td>Taxonomy of object-orientated methodology comparisons</td>
<td>168</td>
</tr>
<tr>
<td>6-13</td>
<td>Number of comparisons for Nm methodologies</td>
<td>169</td>
</tr>
<tr>
<td>6-14</td>
<td>Representing classes and objects</td>
<td>171</td>
</tr>
</tbody>
</table>
Figure 8-15 - Representing SSL as an extension of the GOOKB 218
Figure 8-16 - NSM table for the SSL module modelling language 219
Figure 8-17 - Supporting SSL with MOOT 220
Figure 8-18 - Explain method of the ComplexCritic class in the CKB 223
Figure 8-19 - An example SSL method model 224
Figure 8-20 - Joosten trigger model (Joosten, 1995) 225
Figure I-1 - Dimensions of the evaluation framework 248
Figure IV-1 - Sieve of Eratosthenes version 1 258
Figure IV-2 - Sieve of Eratosthenes version 2 265
Figure VI-1 - The main components of the SSL compiler 278
Figure VI-2 - Representing types in the SSL compiler 280
Figure VI-3 - Statements and expressions in the SSL compiler 281
Figure VI-4 - Representing modules, classes, operations and methods in the SSL compiler 282
Figure VII-1 - Components of the SSL-VM 286
Figure VII-2 - Representing SSL objects and SSL classes 288
Figure VII-3 - The classes involved in processing a message on the SSL-VM 290
Figure VII-4 - Executing a method on the SSL-VM 291
Figure VII-5 - Binding a message to a method on the SSL-VM 292
Figure VII-6 - Implementation of the reference counting garbage collection scheme 294
Figure VII-7 - Implementation of the SSL Instance Proxy class 295
LIST OF TABLES

Table 1-1 - First generation object-orientated methodologies 8
Table 1-2 - Second generation object-orientated methodologies 9
Table 1-3 - History of CASE tools (Ferguson, 1998) 13
Table 2-1 - Four layer meta-modelling architecture 28
Table 2-2 - Meta-CASE tools 40
Table 5-1 - SSL-VM types 147
Table 5-2 - SSL-BC instruction set 148
Table 7-1 - Correspondance between syntax and semantic elements 186
Table 9-1 - Practical work completed during the research 230
Table VII-1 - Implementation of SSL types in the SSL-VM 287
ACKNOWLEDGMENTS

Colours

You reap what you sow. Put your face to the ground.
Here come the marching men. Your colours wrapped around.

_The Sisters of Mercy_

The research detailed in this thesis was supported by two PGSF funded research projects (MAU-503, MAU-807). Financial assistance was also received with a New Zealand postgraduate scholarship.

The following people deserve special mention.

*My supervisors, who inspired me and taught me a great deal*

Assoc. Prof. Daniela Mehandjiska-Stavreva, Prof. Mark Apperley

*The Masters and Honours students who were also involved in this research*

Paul Clark, Steven Adams, Hong Yu, Duane Griffin, Sarisha Dasari, Mi Duk Choi, Jonathan Ham

*Two people that selflessly proof-read the thesis*

Rachel Page, Wendy Browne

*My family, without whom this would not have been possible*

Michael Page, Susan Page, Audrey Isaac, Rachel Page, Jonathon Page, Ruth Page

*My friends (you know who you are), who all helped without knowing it*

Extra thanks to: Nick Earle, Paul Clark, Duane Griffin, Lisabeth Weston, Shamus Smith, Luke Usherwood, Marion Moore, Andrew Turvey, James Fulton, Steven Adams
The content of the glossary has been derived from a range of dictionaries (Collins, 1995; Nuttals, 1902; Readers Digest, 1988; Oxford, 1993; Mirriam-Webster, 1998), the Dictionary of Object Technology (Firesmith and Eykholt, 1995) and (D’Souza and Wills, 1998; Jacobson et al., 1995; Pressman 1997; Schach, 1993, 1997; Somerville, 1996).

**Abstraction.** Any model that includes the most important, essential, or distinguishing aspects of something while suppressing or ignoring less important, immaterial, or diversionary details. The result of removing distinctions so as to emphasise commonalities.

**Arity.** The cardinality of something. For example the arity of a relation specifies the number of concepts that are involved in the relation.

**Attribute.** Any named property used as a data abstraction to describe its enclosing object, class or extent.

**Behaviour.** Anything that an organism does involving action and response to stimulation. The way in which someone behaves; also: an instance of such behaviour.

**Bind.** To place under certain constraints. To cohere or cause to cohere. To place under obligation; oblige.

**Binding.** Any selection of the appropriate method for an operation on receipt of a corresponding message.

**Browser.** Any view that allows you to access hierarchically organised and indexable information.

**CASE Tool.** A) Any computer based tool for software planning, development and evolution. This includes all examples of computer-based support for the managerial, administrative, or technical aspects of any part of a software development project. B) Products that assist the software engineer in developing and maintaining software.

**CASE.** An acronym that stands for Computer Assisted Software Engineering.

**CKB.** Core Knowledge Base. A library of methodology semantic components that implements a meta-model of methodology.

**Class.** Any uniquely identified abstraction (i.e. a model) of a set of logically related instances that share the same or similar characteristics. The combination of a type interface and associated type implementation.

**Classification.** The act of forming into a class or classes; a distribution into groups such as classes, orders, families, etc., according to some common relations or affinities.
Cohesion. The degree, to which something models a single abstraction, localising only features and responsibilities related to that abstraction.

Component. A) Any standard, reusable, previously implemented unit that is used to enhance the programming language constructs and to develop applications. B) An independently deliverable unit of software that encapsulates its design and implementation and offers interfaces to the out-side, by which it may be composed with other components to form a larger whole.

Coupling. The degree to which one thing depends on another. Low coupling is desirable because it produces better encapsulation, maintainability, and extendibility with fewer objects needlessly affected during iteration.

Encapsulation. To enclose in or as if in a capsule; the act of enclosing in a capsule. The physical localisation of features.

Engineering. The application of scientific principles to such practical ends as the design, construction, and operation of efficient, economical structures, equipment and systems. The application of science to the design, building, and use of machines, constructions etc.

GOOKB. Generic Object Oriented Knowledge Base. A library of methodology semantic components that implements a meta-model of concepts germane to all object-orientated methodologies.

Identity. Individuality.

Information Hiding. The deliberate and enforced hiding of information (e.g. design decisions, implementation details) from clients. The limiting of scope so that some information is invisible outside of the boundary of the scope.

Inheritance. The incremental construction of a new definition in terms of existing definitions without disturbing the original definitions and their clients.

Instance. Anything created from or corresponding to a definition.

Interface. The visible outside, user view of something.

Language. Any method of communicating ideas, as by a stream of signs, symbols, gestures or the like. The special vocabulary and usage of a scientific professional or other group. The speech or expression of ideas.

MDT. An acronym that stands for Methodology Description Table. The Methodology Description Table provides an index of the methodologies supported by MOOT.

Message Send. The sending of a message to an object.

Message. Any communication sent or received by an object.

Meta. A Greek prefix signifying beyond, after, with, among and frequently expressing change. Going beyond or transcending. Used with the name of a discipline to designate a
new but related discipline designed to deal critically with the original one. Of a higher or second-order kind.

Meta-CASE Tool. A) A meta-CASE tool is any tool that provides automated or semi-automated support for developing CASE tools. B) … are CASE tools which are used to generate other CASE tools. C) A CASE tool that operates on CASE tools.

Meta-language. The natural language, formal language, or logical system used to discuss or analyse another system. A form of language used to discuss a language.

Method. A) Mode of procedure, logical arrangement, orderly arrangement, system of classification. A means or manner of procedure, especially a regular and systematic way of accomplishing anything. The procedures and techniques characteristic of a particular discipline or field of knowledge. A special form of procedure esp. in any branch of mental activity. B) A way of carrying out a complete phase such as such as design or integration. C) The hidden implementation of an associated operation.

Methodology CASE Tool. A CASE tool that supports one or more software development methodologies and attempts to span most of the software development life-cycle.

Methodology. The science of scientific method of classification. From the Greek method and logis (science). The system of principles, practices, and procedures applied to any specific branch of knowledge. The science of method; a body of methods used in a particular branch or activity.

Model. A) Archetype; a description or analogy used to help visualise something that cannot be directly observed; a system of postulates, data, and inferences presented as a mathematical description of an entity or state of affairs. A preliminary pattern or representation of an item not yet constructed. A tentative framework of ideas describing something intangible and used as a testing device. B) A model clarifies – for a person or group of people – some aspect or perspective on a thing or event.

MOOT. Meta Object Orientated Tool. A new meta-CASE tool developed as a result of this research.

NDL. Notation Definition Language. A new language used to define the syntax of a methodology in MOOT.

Notation. A system of characters, symbols, or abbreviated expressions used in an art or science or in mathematics or logic to express technical facts or quantities.

NSM. An acronym that stands for Notation-Semantic Mapping. NSM tables are used to implement late binding of NDL and SSL methodology descriptions.

Object. Any abstraction that models a single thing.

Operation. Any service that may be requested.

Polymorphism. The ability of a single name to refer to different things having different forms.
Process. A) A system of operations in the production of something. A series of actions, changes or functions that bring about an end or result. A course of action or proceeding, esp. a series of stages in manufacture or some other operation. B) ... the way we produce software. It starts with concept exploration and ends when the product is finally retired. C) ... the set of activities and associated results which produce a software product.

Relation. Connection by consanguinity or affinity; kinship; relationship; as, the relation of parents and children; an abstraction belonging to, or characteristic of, two entities or parts together.

Semantic. Of, or relating to, meaning in language.

Software Development Life-cycle (SDLC). A process by which software engineers build computer applications.

Software Engineering. A) The application of a systematic, disciplined, quantifiable approach to the development, operation, and maintenance of software; that is the application of engineering to software. B) ... is concerned with the theories, methods and tools that are needed to develop software for computers. C) A discipline whose aim is the production of quality software that satisfies the user's needs, and is delivered on time and within budget.

Software Project. A software project consists of a set of models (built using a particular methodology) which collectively define the software being constructed.

SSL. Semantic Specification Language. A new object-orientated language used to define the semantics of a methodology in MOOT.

SSL-BC. SSL Byte Code. A platform independent, binary, representation of SSL, which is generated by the SSL compiler.

SSLC. SSL compiler.

SSL-VM. SSL Virtual Machine. A new virtual machine which supports efficient processing of SSL.

State. Any status, situation, condition, mode, or life-cycle phase of an object or class during which certain rules of overall behaviour (e.g. response to messages) apply.

Syntax. The way in which linguistic elements (as words) are put together to form constituents (as phrases or clauses).

Tool. A thing used in an occupation or pursuit. Any instrument of use or service.

Type. A lower taxonomic category selected as a standard of reference for a higher category. The declaration of the interface of any set of instances (e.g. objects) that conform to this common protocol.
PUBLICATIONS

The results of this research have been presented in the following refereed publications.


