



A Framework for Generating Operational
Characteristic Curves for Semiconductor
Manufacturing Systems using Flexible and
Reusable Discrete Event Simulations

Néill M. Byrne B.E., M.Sc.

This report is submitted in accordance with the requirements of Dublin City University for
the degree of Doctorate of Philosophy in Engineering.

Supervisors: Dr. John Geraghty and Dr. Paul Young

Submitted to the School of Mechanical and Manufacturing Engineering, Dublin City
University, November 2011.

I hereby certify that this material, which I now submit for assessment on the programme of study leading to the award of Ph.D. is entirely my own work, that I have exercised reasonable care to ensure that the work is original, and does not to the best of my knowledge breach any law of copyright, and has not been taken from the work of others save and to the extent that such work has been cited and acknowledged within the text of my work.

Néill M. Byrne
ID: 98463730

April 2, 2012

For the Nana

Acknowledgements

To my supervisors, Dr. Paul Young and Dr. John Geraghty, thank you for all the guidance, support, advice, kindness and encouragement that you provided me.

I also wish to thank my industrial contacts, Mr. Steve Sievwright and Mr. Ken Daly, your expertise in the area was invaluable to the completion of this thesis.

To my family, friends and colleagues, I am indebted to you for keeping me on the straight and narrow throughout this journey.

And finally to Ann, many thanks for your patience, and foregoing many a Sunday activity.

Contents

Acknowledgements	i
Table of Contents	ii
List of Figures	viii
List of Tables	xii
Glossary	xv
Nomenclature	xvii
Kendall Notation	xvii
Acronyms	xviii
Abstract	xx
1 Introduction	1
1.1 Statement	1
1.2 Motivation	1
1.3 Thesis Outline	2
2 Literature Review	4
2.1 Semiconductor Manufacturing	4
2.1.1 Semiconductor industry economics	4
2.1.2 Fabrication plants	9
2.1.3 Operational complexities in a wafer fab	11
2.2 Operating Curves	14
2.2.1 Queueing systems	15
2.2.2 The advantages of using operating curves	19

2.2.3	Factors that impact fab performance	20
2.2.4	Fab utilisation and bottlenecks	21
2.3	Modelling the Fab and its Operating Curve	24
2.4	Generating Operating Curves using Analytical Modelling Methods	27
2.4.1	Modelling semiconductor fabs using analytical models	27
2.4.2	Benchmarking fabs using operating curves	31
2.4.3	Implications of the fundamental assumptions associated with queuing theory	34
2.4.4	Queueing networks	37
2.4.5	Advanced queuing approximations	38
2.5	Discrete Event Simulation Modelling	39
2.5.1	Steps in a simulation study	39
2.5.2	Components of a DES model	45
2.5.3	Justification for using DES modelling	47
2.5.4	Flexible reusable DES modelling	51
2.6	Summary	57
3	An Automated Framework for Designing Discrete Event Simulation Experiments	59
3.1	Simulation Effort	60
3.1.1	Selection and location of design points on an operating curve	60
3.1.2	Allocating simulation effort	62
3.2	Method of Independent Replications	68
3.3	Whitt Simulation Run Length	69
3.4	Methods for Identifying the Initial Bias and Warm Up Period	71
3.4.1	SPC method	73
3.5	Operational Characteristic Surfaces	77
3.6	Summary	78
4	Case Study: A Flexible Toolset Modelling Application	80
4.1	Testbed Background	81
4.2	Front-End for the FTM Application	82
4.3	Data Mining and Collection	84
4.3.1	Determining arrival patterns	86
4.3.2	Determining lot processing patterns	88
4.3.3	Downtime event distributions	90
4.3.4	Lot selection and prioritisation of operations	92
4.3.5	Exporting information to ExtendSim	93

4.4	ExtendSim DES Model	95
4.4.1	Lot Generator block	97
4.4.2	Tool Generator block	98
4.4.3	Unscheduled downtime generator block	99
4.4.4	PM Generator block	99
4.4.5	Pairing block	100
4.4.6	Activity delay paths	103
4.5	Recording Simulation Data from ExtendSim	103
4.6	Generating the Operating Curve	105
4.6.1	Estimating the theoretical operating curve	106
4.7	Model Verification & Validation	108
4.8	IDEF0 Model Interpretation	110
4.9	Summary	115
5	Semiconductor Fab Model A	117
5.1	Semiconductor Wafer Manufacturing Data Format Specification	118
5.2	Project Objectives	119
5.3	Modelling Strategy	120
5.4	Model Input and GUI	123
5.5	Communicating with ExtendSim from VB	124
5.6	Model Description	124
5.6.1	Lots and batching	124
5.6.2	Lot processing	127
5.6.3	Tool downtime	129
5.6.4	Operators and breaks	132
5.6.5	Rework and scrap	133
5.6.6	Capturing the model output	134
5.6.7	Checking model stability	136
5.7	Analysis of Sematech Dataset 1	138
5.7.1	Batch size policies	139
5.7.2	System bottleneck analysis	142
5.7.3	Comparison with the CXFC approximation	146
5.7.4	Downtime	147
5.7.5	Operators	148
5.8	Model Verification & Validation	150
5.9	IDEF Model Diagrams	153
5.10	Summary	159

6	Semiconductor Fab Model B	161
6.1	Justification for the use of Python and SimPy	161
6.2	Model Input and GUI	162
6.3	Modelling Entities and Processes using SimPy	162
6.3.1	Lot and operation PEM's	164
6.3.2	Tool PEM's	165
6.3.3	Downtime PEM's	166
6.3.4	Operator and break PEM's	166
6.4	Capturing Model Output and Displaying Operating Curves	167
6.5	Analysis of the Minifab Dataset	169
6.5.1	Operating curve results for minifab dataset without operators or downtime	170
6.5.2	Operating curve results for minifab dataset with operators and downtime	173
6.6	ExtendSim and SimPy Comparison	175
6.7	Model Verification & Validation	177
6.8	Summary	179
7	Discussion	181
7.1	Overview	181
7.2	Optimum Location of Simulated Design Points on Operating Curves . . .	183
7.3	Operating Points, Curves and Surfaces	184
7.4	Reflections on the FTM application	186
7.5	Craft-based versus Generic Modelling	187
7.6	Industrial Implications	189
8	Conclusion	192
8.1	Technical Contributions	193
8.2	Recommendations for Future Work	193
	References	195
	Appendices	
A	Coded Algorithms for Designing DES Experiments	A-1
A.1	Required number of simulation replications	A-1
A.2	Whitt approximation for simulation run length	A-2
A.3	Batch size approximation	A-2
A.3.1	Von Neumann algorithm	A-3

A.3.2	Anderson-Darling test for normality	A-4
A.4	SPC Algorithm	A-4
A.5	Miscellaneous Functions	A-7
A.5.1	Inverse normal distribution function	A-7
A.5.2	Batch means method	A-7
A.5.3	Batch variance	A-8
A.5.4	Queue operating point	A-9
B	Flexible Toolset Modelling Application Code	B-1
B.1	Front-End	B-1
B.2	Data Collection and Sorting	B-3
B.2.1	Data pull and cross-referencing	B-3
B.2.2	Calculating arrival rates	B-5
B.2.3	Calculating process time	B-6
B.2.4	Estimate downtime parameters	B-8
B.2.5	Lot selection parameters	B-12
B.3	VBA Wrapper for ExtendSim	B-18
B.4	ExtendSim Custom Blocks	B-23
B.4.1	Lot generator code	B-23
B.4.2	Tool generator code	B-28
B.4.3	Unscheduled downtime generator code	B-32
B.4.4	Preventative maintenance generator code	B-36
B.4.5	Pairing block code	B-41
B.5	Post Processing Scripts	B-50
C	Semiconductor Wafer Manufacturing Data Format Specification	C-1
C.1	File Description Overview	C-2
C.2	File Descriptions	C-2
C.3	Additional Information	C-7
D	Code for Fab Model A	D-1
D.1	Simulation Model Inputs	D-1
E	Code for Fab Model B	E-1
F	Johnson Distribution	F-1
F.1	Algorithm	F-2
F.2	Software Interpretation of Johnson Distribution	F-4
F.3	Python Implementation	F-4

F.4	VB Implementation	F-7
G	Verification and Validation Techniques	G-1
G.1	Verification Techniques	G-1
G.2	Validation Techniques	G-4
H	Publications	H-1

List of Figures

2.1	Increase in global semiconductor sales from 1993 to 2010, <i>data taken from Worldwide Sales of Semiconductors in Billion USD</i> (2010).	5
2.2	Transistor counts for integrated circuits and their dates of introduction (Simon, 2008)	6
2.3	Operating curve for an M/M/1 queueing system showing the relationship between x-factor and utilisation.	18
2.4	Typical inflection region on an M/M/1 queue operating curve.	20
2.5	System modelling methods, <i>based on</i> (Gordon, 1977; Law and Kelton, 1997).	26
2.6	Locating the current operating point on an operating curve generated by an analytical queueing model.	28
2.7	Steps in a simulation study.	41
2.8	Phases of a simulation study.	42
2.9	Components of a semiconductor fab categorised using simulation model components suggested by Jeong et al. (2009).	47
2.10	Modelling effort and model reusability.	53
3.1	Operating curve indicating the simulation output variance.	61
3.2	M/M/1 operating curve showing the ‘vertical’ and ‘horizontal’ asymptotes and the curved area of interest.	63
3.3	The curvature level $k(u)$ for an M/M/1 queueing approximation according to Eq.(3.4).	64

3.4	Operating curve and equivalent u/CT curve for M/M/1 queue, showing the u/CT region and an optimum operating point at $u = 0.5$	66
3.5	Design points for an M/M/3 queuing system at 100%, 95%, 80% and 40% of u/CT	67
3.6	Recommended run length for G/G/1 queue with moderate variability. . .	70
3.7	Batched time series transient and statistical process control (SPC) control parameters for an M/M/1 queue, showing failure of Test 1 at \bar{Y}_{36}	76
3.8	Operating Surface for a G/G/1 queueing system according to Eq.(2.4). .	77
3.9	u/CT Surface for a G/G/1 queueing system.	78
3.10	Summary flow chart for the automated framework for designing discrete event simulation (DES) experiments.	79
4.1	Tool/toolset selection for the Flexible Toolset Modelling (FTM) application.	83
4.2	Selection of experimental parameters for simulation model.	84
4.3	Combined database time stamps for a single process tool.	85
4.4	Arrival histogram and exponential fit for lots requiring operation A on ‘H’ toolset.	86
4.5	Arrival histogram and exponential fit for lots requiring operation B on ‘H’ toolset.	87
4.6	Arrival histogram and exponential fit for lots requiring operation C on ‘H’ toolset.	87
4.7	Unbounded Johnson distribution fit for lots requiring operation A on tool T2.	88
4.8	User prompt to distinguish between scheduled and unscheduled downtime events recorded in the tool history database.	91
4.9	Creation of preventative maintenance (PM) schedules through a graphical user interface (GUI) wizard.	91
4.10	Visual Basic (VB) Userform used to select lot prioritisation options for each tool.	92
4.11	VB Userform used to rank processing priority for operations.	93
4.12	Traditional job-driven graphical modelling approach.	95
4.13	Screenshot of ExtendSim model used by the FTM application.	96
4.14	Dialog of the custom Lot Generator block used for the FTM application.	97
4.15	Dialog of the custom Tool Generator block used in the FTM application.	99
4.16	Dialog of the custom unscheduled downtime generator block used in ExtendSim.	100
4.17	Dialog of the custom PM Generator block used in ExtendSim.	100
4.18	Logic code execution for Pairing block used in the FTM application. . . .	101

4.19	Details of lots residing in the Pairing block during runtime.	102
4.20	Flow system for flexible simulation model used in the FTM application. .	104
4.21	The operating curves generated by the simulation and the equivalent M/G/m queueing approximation for the system.	105
4.22	Tool operating points from historical records plotted alongside the queue- ing approximation and simulated operating curves.	110
4.23	IDEF0 standard adapted to the viewpoint of the modeller.	111
4.24	Overview IDEF0 diagram (A-0) for FTM ExtendSim model.	111
4.25	‘Run Simulation’ (A0) IDEF0 diagram for FTM ExtendSim model. . . .	112
4.26	‘Generate Lots’ (A1) IDEF0 diagram for FTM ExtendSim model.	112
4.27	‘Generate Tools and Downtime’ (A2) IDEF0 for FTM ExtendSim model.	113
4.28	‘Pair Items’ (A3) IDEF0 diagram for FTM ExtendSim model.	113
4.29	‘Service’ (A4) IDEF0 diagram for FTM ExtendSim model.	114
4.30	‘Repair’ (A5) IDEF0 diagram for FTM ExtendSim model.	114
4.31	IDEF1x diagram showing the objects and attributes used to describe the entities in the FTM ExtendSim model.	115
5.1	Modelled entities and their attributes described using IDEF1x.	122
5.2	Dialog option available for user to select a dataset.	123
5.3	Additional release pattern options for selection.	123
5.4	Portion of time the operator is occupied during the process step.	127
5.5	Cycle time trace of both products in Sematech dataset 1.	137
5.6	Operating curves for dataset 1 using a minimum and maximum batch sizing policy.	141
5.7	Comparison of M/D/m approximation for bottleneck toolset TS67 and fab operating curve predicted by simulation model.	144
5.8	Comparison of results for flexible reusable model, x-factor and complete x-factor contribution (CXFC) produced by Delp et al. (2006).	146
5.9	Comparison of operating curves for dataset 1 with unreliable machines according to Table 5.11 and Table 5.17.	149
5.10	Operating curve for dataset 1 using operators under a minimum and max- imum batching policy, according to Table 5.18.	150
5.11	Overview IDEF0 diagram (A-0) for ExtendSim model.	154
5.12	‘Run simulation’ (A0) IDEF0 diagram for ExtendSim model.	155
5.13	‘Generate lots’ (A1) IDEF0 diagram for ExtendSim model.	155
5.14	‘Generate tools’ (A2) IDEF0 diagram for ExtendSim model.	156
5.15	‘Generate operators’ (A3) IDEF0 diagram for ExtendSim model.	156
5.16	‘Pair items’ (A4) IDEF0 diagram for ExtendSim model.	157

5.17	‘Service’ (A5) IDEF0 diagram for ExtendSim model.	157
5.18	‘Operator queue’ (A6) IDEF0 diagram for ExtendSim model.	158
5.19	‘Testing’ (A7) IDEF0 diagram for ExtendSim model.	158
5.20	‘Repair’ (A8) IDEF0 diagram for ExtendSim model.	159
6.1	Visual interpretation of <i>Semiconductor Wafer Format Specification</i> in MySQL database.	163
6.2	GUI for the Python/SimPy application.	164
6.3	Visualisation of output data from simulation model stored in MySQL database.	168
6.4	Output GUI for SimPy model.	168
6.5	Process step-centric representation of minifab dataset.	170
6.6	Standard operating curve and u/CT curve for minifab dataset without operators or downtime.	171
6.7	Comparison between x-factor results for simulation, M/M/1 and M/D/1 approximation.	172
6.8	Operating curves for minifab dataset with operators and downtime.	173
6.9	Comparison of simulated operating curves for ExtendSim and SimPy models using the minifab dataset.	176
7.1	Operating surface for G/G/10 queue with lot dedication restriction.	186
B.1	User prompt to distinguish between scheduled and unscheduled downtime events recorded in the tool history.	B-12
B.2	VB Userform used to select lot prioritisation options for each tool.	B-13
B.3	VB Userform used to rank processing priority for operations.	B-16
B.4	Dialog of custom Lot Generator block for FTM Application.	B-23
B.5	Dialog of custom Tool Generator block for FTM Application.	B-28
B.6	Dialog of custom unscheduled downtime generator block for FTM Application.	B-33
B.7	Dialog of custom preventative maintenance generator block for FTM Application.	B-37
F.1	An example of the empirical data and the fitted Johnson unbounded frequency distribution using the algorithm described in this chapter.	F-5

List of Tables

2.1	Justification for using simulation modelling to generate operating curves in semiconductor manufacturing, based on the recommendations offered by Banks and Gibson (1996, 1997a).	50
3.1	Simulation run length approximation for G/G/1 queueing system according to the Whitt estimator (Whitt, 1989b).	70
3.2	SPC test results for M/M/1 queueing system.	76
4.1	Time-stamp sources.	85
4.2	Johnson distribution parameters for each operation on each tool in toolset ‘H’.	89
4.3	Average post-processing waiting time of all lots on each tool.	90
4.4	Information required by ExtendSim simulation model.	94
4.5	Attributes used by ExtendSim model items.	98
4.6	Model time stamps recorded during runtime.	104
4.7	Utilisation and cycle time predicted by the simulation model.	106
4.8	Techniques used to verify the FTM application.	108
4.9	Techniques used to validate the FTM application.	109
5.1	Data files used for wafer data format specification.	118
5.2	Comparison of the traditional use of basic simulation objects and an entity-centric approach to model a semiconductor fab.	120
5.3	Entity-centric approach to modelling semiconductor fabs.	121

5.4	VB wrapper functions for ExtendSim.	125
5.5	Degree of re-entrancy for <i>Semiconductor Wafer Manufacturing Data Format Specification</i> sample datasets.	126
5.6	Description of <i>LotTrace</i> database for collecting model output.	135
5.7	Description of <i>ToolTrace</i> database for collecting model output from the tools.	136
5.8	Description of <i>OperatorTrace</i> database for collecting model output from the operators.	136
5.9	Description of Sematech dataset 1 from <i>MASM Lab Factory Datasets</i> (1996).	138
5.10	Sample run of Sematech dataset 1 using Factory Explorer.	139
5.11	Simulation model results for Sematech dataset 1 with a maximum batch size policy and no operators, downtime or rework.	140
5.12	Simulation model results for Sematech dataset 1 with a minimum batch size policy and no operators, downtime or rework.	140
5.13	The approximate average arrival rate permissible for the system and for the bottleneck toolset TS67, given that $m = 7$	143
5.14	Operation details for TS67.	144
5.15	Comparison of results for flexible reusable model, simulation model produced by Delp et al. (2006), and the CXFC approximation.	147
5.16	Unreliable toolsets in dataset 1 ranked by least availability.	148
5.17	Simulation model results for dataset 1 with unreliable machines.	148
5.19	Techniques used to verify the ExtendSim model.	150
5.18	Simulation model results for dataset 1 using operators with a minimum and maximum batch sizing policy.	151
5.20	Comparison of reported, calculated and simulated raw process times for Products 1 and 2 for dataset 1.	152
5.21	Techniques used to validate the ExtendSim model and application.	153
6.1	Description of Sematech minifab dataset from <i>MASM Lab Factory Datasets</i> (1996).	169
6.2	Comparison of reported, calculated and simulated raw process times for minifab dataset.	170
6.3	Cycle time and x-factor results for Sematech minifab dataset with no operators or downtime.	171
6.4	Cycle time and x-factor results for Sematech minifab dataset with no operators or downtime.	174
6.5	Average availability for minifab toolsets.	175
6.6	Comparison of simulation model results for ExtendSim and SimPy models.	176

6.7	Techniques used to verify the SimPy model.	177
6.8	Techniques used to validate the SimPy model.	179
C.1	Data files used for wafer data format specification.	C-2
C.2	Structure of PROCESS ROUTE (<i>pr</i>) file.	C-3
C.3	Structure of REWORK SEQUENCE (<i>rw</i>) file.	C-4
C.4	Structure of TOOL SET (<i>ts</i>) file.	C-5
C.5	Structure of OPERATOR SET (<i>os</i>) file.	C-6
C.6	Structure of VOLUME RELEASE (<i>vr</i>) file.	C-6
C.7	Comparison of Sematech datasets.	C-9
D.1	Input data subroutines and functions for the Sematech model.	D-2
F.1	Parameters for estimation of Johnson bounded and unbounded distribution.	F-3
F.2	Parameters for estimation of Johnson lognormal distribution.	F-4
F.3	Different interpretation of Johnson distribution parameters.	F-5
G.1	Verification techniques for simulation modelling (Whitner and Balci, 1989).	G-2
G.2	Validation techniques for simulation models (Sargent, 1998).	G-4

Glossary

Many of the descriptions given are based on definitions from Hopp and Spearman (2001), with minor adjustments to some of the terminology with regards to semiconductor manufacturing.

availability the fraction of uptime at a station, tool or toolset.

batch a grouping of lots.

cycle time the average time from when a lot or job is released into a system to when it exits (hrs).

fab a shortened industry term for a semiconductor wafer fabrication facility.

inter-arrival times average time between arrivals to a system (hrs).

lot a grouping of wafers that travel as a single unit.

mean effective process time average time required to do a job, including all production detractors (such as setups and downtime) but not including time that a system is starved for lack of work or blocked by busy downstream systems (hrs).

mean time before failure the mean uptime between successive failures (measured from the end of the last failure to the beginning of the next failure) of a machine or tool (hrs).

mean time to repair the mean downtime measured as the mean amount of time taken to repair a machine or tool (hrs).

operating curves short for operational characteristic curves, a curve defining the cycle time and utilisation relationship of a system.

process time the amount of time taken to complete a job or task at a station or tool (hrs).

raw process time the sum of the process times of a routing or system (hrs).

throughput the average output of a system (lots per hour).

tool a semiconductor industry term for a piece of machinery or equipment.

utilisation the fraction of time a system is not idle for lack of WIP.

variability the non-uniformity of a class of entities.

work in process the number of units of inventory between the start and end points of a system (number of lots).

General subscript conventions:

- subscript **a** indicates a parameter that describes the arrival pattern to a system.
- subscript **e** indicates a parameter that describes “effective” process times of a system which includes production detractors.
- subscript **f** indicates a parameter that describes the average time between successive failures at a station or tool.
- subscript **r** indicates a parameter that describes the average repair or maintenance time at a station or tool.
- subscript **0** indicates a parameter that describes the natural time of a process without any production detractors.

Kendall Notation

Kendall notation is a method for describing single stage queueing systems using the syntax,

$$(a/b/c) : (d/e/f)$$

where the placeholders a - f refer to,

- a:** arrival distribution,
- b:** service (or process) distribution,
- c:** number of parallel servers,
- d:** queue discipline (*default is FIFO*),
- e:** number of customers allowed in the system (*default is unlimited*),
- f:** maximum number of customers that can be called from (*default is unlimited*).

The arrival and service distribution patterns use further notation to designate the distribution type,

- D:** deterministic uniform distribution,
- M:** exponential distribution, also known as a Markovian distribution,
- G:** a general distribution including; lognormal, normal, beta, etc.,
- E:** specific cases of the Erlang distribution,

Acronyms

AMHS	automated material handling system
CMSD	Core Manufacturing Simulation Data
CXFC	complete x-factor contribution
DES	discrete event simulation
DOE	design of experiments
DoR	degree of re-entrancy
EPT	effective process time
FIFO	first in first out
FOUP	front opening unified pod
FTM	Flexible Toolset Modelling
GUI	graphical user interface
IC	integrated circuit
IDE	integrated development environment
KPI	key performance indicator
LACTE	load-adjusted cycle time efficiency
MIMAC	Measurement and Improvement of Manufacturing Capacity
MTBF	mean time before failure
MTTR	mean time to repair
MWBF	mean wafers before failure
OEE	overall equipment effectiveness

PEM	process execution method
PM	preventative maintenance
RPT	raw process time
SME	subject matter expert
SPC	statistical process control
SQL	Structured Query Language
TOC	theory of constraints
UML	unified modelling language
VB	Visual Basic
VBA	Visual Basic for Applications
WIP	work in process
XML	extensible markup language

A Framework for Generating Operational Characteristic Curves for Semiconductor Manufacturing Systems using Flexible and Reusable Discrete Event Simulations

Néill M. Byrne

This thesis proposes a framework for generating operating curves for semiconductor manufacturing facilities using a modular flexible discrete event simulation (DES) model embedded in an application that automates the design of experiments for the simulations. Typically, operating curves are generated using analytical queueing models that are difficult to implement and hence, can only be used for benchmarking purposes. Alternatively, DES models are more capable of capturing the complexities of a semiconductor manufacturing facility such as re-entrancy, rework and non-identical toolsets. However, traditional craft-based simulations require much time and resources. The proposed methodology aims to reduce this time by automatically calculating the parameters for experimentation and generating the simulation model. It proposes a novel method to more appropriately allocate simulation effort by selecting design points more relevant to the operating curve.

The methodology was initially applied to a single toolset model and tested as a pilot case study using actual factory data. Overall, the resulting operating curves matched that of the actual data. Subsequently, the methodology was applied to a full semiconductor manufacturing facility, using datasets from the *Semiconductor Wafer Manufacturing Data Format Specification*. The automated framework was shown to generate the curves rapidly and comparisons against a number of queueing model equivalents showed that the DES curves were more accurate. The implications of this work mean that on deployment of the application, semiconductor manufacturers can quickly obtain an accurate operating curve of their factory that could be used to aid in capacity planning and enable better decision-making regarding allocation of resources.

Introduction

1.1 Statement

This thesis examines the methods for generating fast, accurate and reliable operational characteristic curves for semiconductor manufacturing facilities. A framework for generating these curves is proposed, which consists of a computer application that utilises analytical approximations and statistical methods to generate automated discrete event simulation models.

1.2 Motivation

Understanding and monitoring the efficiency of a semiconductor manufacturing facility in a holistic manner is a difficult activity when one considers the plethora of performance indicators that can be examined. Often, implementing improvements at the facility based

on a single or incorrect performance indicator can lead to a dis-improvement in another performance indicator. Therefore, it is necessary to establish a ‘catch all’ performance indicator that is representative of true factory performance and relevant to the conflicting goals of the factory; a reduction in production lead time and a maximisation of equipment utilisation. An operational characteristic curve (known simply as an operating curve) is capable of capturing these conflicting goals and enabling management and engineers understand how the factory will react to different loading levels, and to aid them in better decision making with regard to capacity planning and resource allocation. However, operating curves are only as accurate as the methods and techniques used to create them. The modelling technique used, has a large impact on the resultant curve and hence, can have significant implications on the decisions based around the curves. Therefore, this thesis examines and evaluates the best possible methods for generating the curve, and ultimately proposes a framework for automating discrete event simulations that can generate fast and reliable curves.

1.3 Thesis Outline

The literature review chapter begins by describing the semiconductor industry and how this commodity-based industry and its driving forces are pushing semiconductor fabrication facilities (known simply as *fabs*) to increase efficiencies and throughput at an increasingly fast rate while reducing lead times. A brief description is given of the inhospitable environment of a semiconductor fab and its complexities, with a particular emphasis on the operational complexities that make managing and controlling a fab a very difficult task, and hence, make modelling and describing the fab an equally complex task. The chapter then goes on to describe how operating curves can be useful for making informed decisions and how they are indicative of the behaviour of such a system. The discussion then moves to focus on the best possible methods for deriving these curves with an emphasis on the two most common methods; analytical approximations and dis-

crete event simulation models. The pros and cons of each method are evaluated and a conclusion is drawn that analytical models are insufficient for a number of reasons and that simulation modelling is the best possible method for capturing fab complexities and the most likely technique of generating reliable operating curves. A case is then made to overcome the difficulties encountered when performing simulation modelling project, by proposing a framework that attempts to automate the process and use analytical rough-cut models to prime a generic discrete event model. Some literature pertaining to the creation and automation of simulation models is then included.

The first development chapter (Chapter 3) introduces the reader to the proposed framework by using best practice methods for designing simulation experiments. These include; establishing the required number of simulation replications, the simulation run length, estimations of steady state and initial bias for deletion. Also included is a novel method for establishing the most appropriate and pertinent location of design points on the operating curve for experimentation in the simulation model.

Chapter 4 details the first flexible, reusable, automated model used for deriving operating curves for single semiconductor toolsets, known as the Flexible Toolset Modelling (FTM) application. A case study of its deployment and implementation at a real semiconductor fab is given.

Chapters 5 and 6 describe a full factory model that utilises a proposed modelling strategy implemented using two different platforms; an ExtendSim simulation model encapsulated in a Visual Basic (VB) application and a SimPy simulation model integrated in a Python based application. The chapters show a comparison between the two implementations and compare the resulting operating curves to a number of analytical approximations discussed in the literature review chapter.

Finally, the discussion (Chapter 7) and conclusions (Chapter 8) summarise the findings and outline the contributions of the thesis.

Literature Review

2.1 Semiconductor Manufacturing

Much of the discussion in this thesis involves the modelling of semiconductor manufacturing systems, and all experimentation and resulting observations were carried out using semiconductor factory datasets. Therefore, this section delivers a general introduction to semiconductors and the related global industry. There is also a discussion on some of the main complexities and components associated with semiconductor manufacturing (Section 2.1.3), as well as a general overview of the economic factors that drive the industry.

2.1.1 Semiconductor industry economics

For many years the largest industry in the world was the automobile industry. This changed at the beginning of the 21st century when the electronics industry surpassed

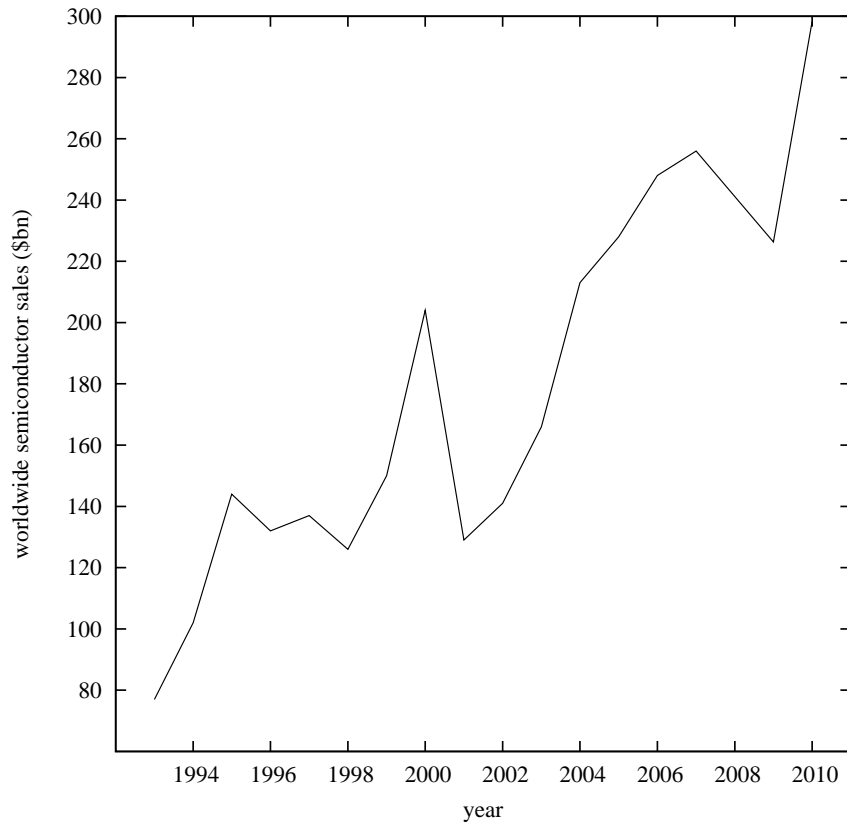


Figure 2.1: Increase in global semiconductor sales from 1993 to 2010, *data taken from Worldwide Sales of Semiconductors in Billion USD (2010)*.

it in terms of global sales (May and Sze, 2004). Much of the electronics industry is based on semiconductor sales, which itself has seen significant growth over the past 15 years (Fig. 2.1), mainly as a result of the growing demand for the integrated circuit (IC) chips built using semiconductors. In fact, most other industries including the aerospace, communications, consumer electronics and automobile industry, rely heavily on IC chips, and in many ways the semiconductor is a fundamental cornerstone of global technological advancement.

More recently, the industry reported global sales in 2009 of \$226 billion and provided over 207,000 jobs in the U.S. (*Semiconductor Industry Association Factsheet*, 2010). Even against the backdrop of the economic downturn in 2008, and a dip in semiconductor sales, the industry rebounded in 2010 with an almost 37% increase in sales over similar figures for 2009. Most of this was due to increased demand for emerging consumer goods such as hand-held devices, tablet PCs, netbooks, smartphones, solid-state hard drives and

high-definition televisions (Ford, 2010).

Despite such positive growth figures, the increase in global sales has also been a source of great economic pressure on chip manufacturers. The constant demand for faster and cheaper chips is driving very competitive chip costing. The rate of advancement of electronics is reducing product life cycles, which in turn, is putting semiconductor manufacturers under increasing competitive pressures. A trend that currently shows little signs of abating according to Abadir (2007). In order to remain competitive, chip manufacturers must bring new advanced products to market in a consistent cyclical manner to offset the rapid devaluation of older products (Hutcheson, 2000).

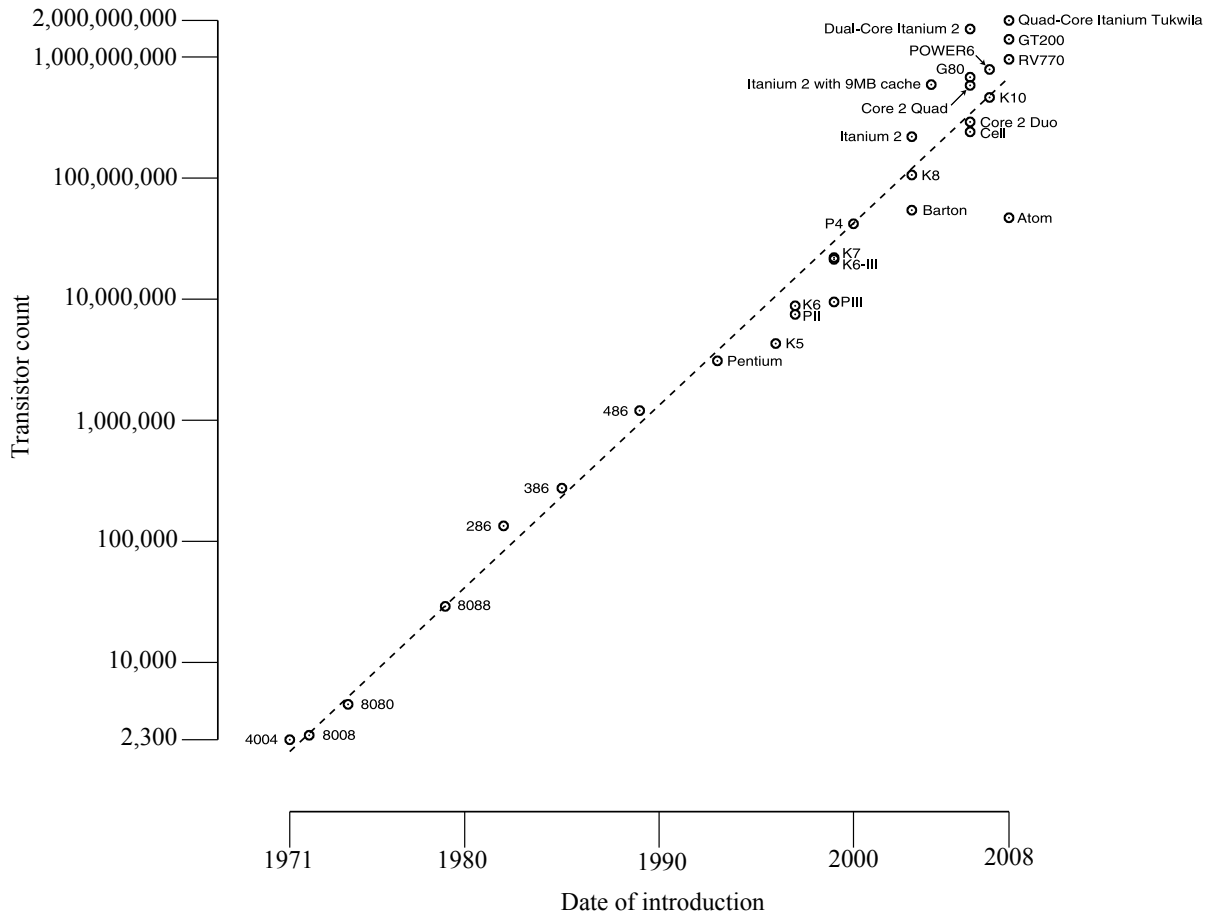


Figure 2.2: Transistor counts for integrated circuits and their dates of introduction. *Edited and reproduced from (Simon, 2008) under the GNU Free Documentation License.*

Moore's law

Technological advancements in the process of creating microprocessors means that it is now possible to fit billions of transistors onto semiconductor chips. The more transistors that can be placed onto a chip, the more powerful the microprocessor.

This rate of evolution of microprocessor was first predicted by Moore in 1965 and is commonly known as Moore's law (Moore, 2000). As can be seen from Fig. 2.2, Moore's law observes that the transistor density that can be inexpensively placed on a semiconductor chip doubles approximately every two years. Such rapid progress has led to an explosion in computing power over the last three decades. Commercial chips have gone from thousands of transistors to tens of billions of transistors per chip with no immediate signs of a slow-down in the rate of advancement.

In fact, the rate at which semiconductor chips are advancing is considered to be a self fulfilling prophecy. Semiconductor manufacturer's targets and efficiencies are now being driven by predictions from Moore's law. The result is that manufacturers are applying innovative solutions and improving their products and manufacturing equipment at a rate that is unmatched by any other industry (Schaller, 1997).

The term *law*, when used to refer to Moore's law can be a bit misleading; it is more a rule of thumb or observation about the phenomenal rate of advancement of semiconductors and the technology required to manufacture them. Some subject matter experts (SMEs), including Moore himself, believe that the advancements which have given rise to the 'exponential-like' increase in processing power also have some inherent limiting factors that will prohibit the future rate of advancement (Moore, 1995). For example, Kwon (2007) and Rupp and Selberherr (2010) believe that it may be economic factors that will halt industry expansion, whereby building and equipping semiconductor manufacturing plants will no longer be feasible due to the increasing cost of equipment and facilities. Similarly, Christensen et al. (2008) painted a picture of semiconductor manufacturing plants as 'unwieldy' factories that required about \$3 million US dollars of product output every day to amortise an initial \$5 billion capital investment over five years. They stated

that covering such a level of investment was almost impossible due to the “incessant cycles of investment and obsolescence that keeps Moore’s law on the march”.

Aside from these economic limits, Moore himself, believes that the trend may be limited due to the physics and dimensions of an electron. A physical limiting barrier may be reached (*circa* 2020); whereby electrons are simply too large, relative to the channel widths of the transistor (Dubash, 2005). This would result in a physical barrier and prohibit further advancement of chip size minimisation.

It is worth noting that these commentators only considered how physical and economic limits may be reached if current trends are left alone to continue. Moore’s Law has been facilitated by various paradigm shifts and new modes of thinking over the history of the semiconductor and it is likely that further new paradigms will materialise. Other ideas, besides chip shrinkage have already been mooted; including integration of computing systems, thus moving towards a ‘single chip’ system (Bai, 2009), or more exotic solutions such as 3D chips (Vucurevich, 2008). Already, the first generation of these 3D chips is due for production and release to the market in early 2012 (Poeter and Hachman, 2011).

Some other SMEs feel that the next step in the advancement of Moore’s law will come with a complete reconfiguration of the semiconductor industry to a more flexible concurrent design and manufacturing system (Bhavnagarwala et al., 2010). The current eco-system of semiconductor development and production consists of a number of somewhat specialised groups (e.g. software, equipment manufacturing, foundry and design) that tend to operate as individual entities. One possible solution is to integrate these services and operations so that an encompassing group is responsible for all aspects of bringing a successful semiconductor product to market. With such a system, semiconductor products could be brought to market faster and have a longer lifecycle. In summary, it appears that while the main facilitator of Moore’s Law, chip minimisation, might be reaching its limit, there are many more avenues awaiting exploitation.

2.1.2 Fabrication plants

Processing of integrated circuits is carried out in a factory that typically consists of three levels; the subfab which is the bottom level, the cleanroom which is the middle level and the air-handling level on the top. The cleanroom is where the wafers are processed. The bottom level; the subfab, allows maintenance access to all the electrical, chemical and fluid conduits that service the cleanroom. The top level controls the airflow into and from the cleanroom.

Fab layout

Generally there are two main categories of manufacturing; product-based (flow type) and process-based (job-shop type) (Kumar, 1994). Product-based is usually used to describe a manufacturing system where the product takes some single line route through the factory and is operated on by workstations along the line. A good example of this style of manufacturing is the automobile industry, where large volumes of low mix or single products tend to move along dedicated product lines and only visit each workstation once.

Alternatively, a job-shop manufacturing system is one where each job has a predefined list of required operations and the parts or widgets must travel along a network of inter-connections between machine groups. This type of system is suited best to factories that produce low volumes with a high product mix. Extremes of this type include prototype manufacturers and traditional tool makers.

Wafer fabs do not appear to fall perfectly into either category of job shop or flow type. They are similar to job shop types in that lots/jobs have a list of operations that must be performed by various machine groups around the plant; however, the operations manifest is far more precise and structured than that of a typical job shop.

Similarly, they are not exactly flow type systems either. The flow of any particular job is defined by the recipe/operations manifest, meaning that there are many crossing flow lines formed by the various product types travelling around the fab. The system becomes

like a network of destinations with overlapping routes. Furthermore, there are many situations where lots revisit some machine groups several times, for follow-up operations.

On a structural level, the most common fab design is a *bay* and *chase* configuration, which are separated by a wall that contains the tools (an industry term for machines) and fab equipment. Tools are loaded from the bay side, and the chase side is used for maintenance access. The chase area is less prone to contaminants than the bay area, so is generally less regulated and subject to less strict cleanroom standards.

The bays typically contain a collective of similar tools. The machines or tools are grouped by similarity of process, and this approach allows operators and equipment to specialise in the manufacturing processes or operations they perform. It also increases the available level of technical support and takes full advantage of the benefits of tool standardisation, an important aspect of manufacturing systems with equipment that is subject to strict operational parameters.

Lot and wafer travel

The silicon wafers that circulate the fab are usually grouped into lots and housed in some form of transportation container. In 200mm wafer fabs, the wafers are placed in cassettes that are housed within a lot box. In the more advanced 300mm fabs, the wafers are held in front opening unified pods (FOUPs). The advantage of a FOUP over a lot box, is that they are sealed environments and only opened within the tools. This helps to keep the wafers protected from contamination and unnecessary handling.

Moving the lots around the fab is a complex task. Most fabs rely on some sort of automated material handling system (AMHS) to move the lots between bays. These AMHSs consist of a moving robot vehicle that operates on a network of tracks, and can both pick and drop FOUPs or lot boxes. There are two main networks of AMHS; intrabay and interbay. The interbay network moves lots between the main stockers. These stockers or buffers hold the lots until they can be processed on the tools in that stocker's bay. They also hold the post-processed lots until the interbay AMHS robot returns to collect

the lot and move it on for the next operation. Within each bay, an intrabay network operates by moving lots from the stocker to the tools and returning them once processing is complete.

2.1.3 Operational complexities in a wafer fab

A wafer fab is an environment with many different aspects that need to be controlled. Controlling all of these aspects often involves finding a trade-off between the competing forces within the fab. The number of aspects that cause a fab to deviate from optimum control are so plentiful that there requires an n-order equivalent number of control policies and operational protocols that must be set in place. The following section discusses some of the most common complexities in the fab.

Tool diversity

Semiconductor fabrication requires a broad range of complex machinery and tools. Classification of tools can be done based on their operation, e.g., photolithography, diffusion, ion implantation, etching, etc. However, with respect to the content of this thesis the actual chemical/physical operation is of little significance, whereas the movement of lots and wafers through the equipment is far more important.

An example of complex machinery in semiconductor manufacturing is that of cluster tools. Cluster tools are typically comprised of several wafer processing modules, managed by a centralised control system that moves the wafers between the modules. Other complex tools have multiple exterior lot-loading ports that can feed individual wafers into the tool.

Re-entrancy

Re-entrancy is a result of the complexity of product flow in the fab. Transistor layers are created on the wafer by performing operations that build up subsequent layers of microscopic conduits and interconnections. These layers are referred to as mask layers

and consist of a defined sequence of operations or processes that are often carried out by different tool groups in the fab. Once a mask layer is complete, the next mask layer may repeat a similar sequence of operations; thus, requiring the wafer to return to the same tool groups again. This process is known as re-entrancy and causes many workflows within the fab to overlap.

Lot and tool dedication

Re-entrancy refers to lots returning to toolsets visited at a previous step, for a follow up operation. The inference here is that any tool within the toolgroup can perform the subsequent operation. In reality, this is not always possible and often the lot can only be processed by the same tool that carried out the prior operation. This is known as *lot-to-tool* dedication and is normally only at photolithography steps.

This dedication is required to increase the wafer yield. During photolithography it is necessary to have precise alignment between critical layers on the wafer. Therefore, to ensure the best possible alignment, the wafer is returned to the tool that performed its previous critical layer. Then any wafer imperfections caused by the tool are carried through to the next layer in the same spot on the wafer. This means that the imperfection or error is contained in the one vertical plane of the wafer, which in effect, increases its yield probability.

Time-critical processes

Time-critical processes are jobs or operations that must be carried out within a specified time window otherwise a yield loss may occur. The time window is usually defined by the current time and the time of the previous operation. If the critical operation does not occur before a designated time then the previous processing step may need to be repeated.

Scheduling

Where there is competition for resources, the optimal scheduling and use of those resources is critical. In a wafer fab there is competition for many resources including AMHSs, machines, tools, operators and technicians. This competition for resources makes proper and efficient scheduling of those resources an important attribute of a well managed wafer fab. Send-aheads, time-critical processes, priority lots (hot lots), setup-avoidance measures and batching rules mean that there is a constant need for dynamic scheduling in the fab.

Downtime and maintenance

The downtime strategy in the fab has a large impact on the efficiency of the overall system. Preventative maintenance (PM) schedules are employed in an effort to avoid any unscheduled downtime or machine failure. A trade-off is usually required to control the level of unscheduled downtime. To do this, the frequency of PMs are regulated carefully; too many may result in unnecessary PM downtime, and too little could result in frequent unscheduled downtimes (Yao et al., 2004).

Scheduling of maintenance periods is also critical to maintain consistent availability of toolsets. Maintenance can also be scheduled based on monitoring tool performance values. For example, some semiconductor manufacturing equipment have very precise operational limits and are continually monitored to ensure they do not fall outside specified control boundaries. If a tool parameter is seen to be deviating out of control a PM task may be brought forward and performed on the tool in the hope that it prevents a possible forthcoming outage.

Rework

After certain process stages or operations, the wafer may be tested. If the test fails it may be necessary to rework the wafer. This might involve an operation to remove a substrate of the wafer, and return the still viable wafer back to a previous step. This means that

a certain percentage of lots will be recirculated back through their workflow or process path.

Some individual wafers that failed inspection tests may even be separated from their lot to be reworked before merging back with the parent lot at a later stage. This is necessary to avoid expensive wafer waste but has the downside of increasing the workflow traffic.

Variability

Variability is described as “the quality of non-uniformity of a class of entities” (Hopp and Spearman, 2001) and refers to the extent of the deviation from the mean. All of the aforementioned sources of complexity such as dedication, rework, setup avoidance and preventative maintenance can all be viewed as sources of flow variability.

2.2 Operating Curves

An observation made by Ahmad and Dhafir (2002) is that performance metrics “ought to be made in light of the company’s strategic intentions which will have been formed to suit the competitive environment in which it operates and the nature of the business”. To summarise, performance metrics should reflect the key objectives of the company and relate to the driving forces that control the industry. Assessing this statement in the context of the semiconductor industry, the most influential driving force is the rapid advancement of semiconductor technology (see Section 2.1.1 on Moore’s Law). A fab’s ‘strategic intentions’ are to manufacture low cost, high performance chips while minimising production lead time and cost of manufacture (Abadir, 2007; Kwon, 2007; McIntosh, 1997).

Based on these factors, it appears that a good fab metric is one that shows cycle time (lead time) and fab cost. Semiconductor fabs typically cost in the billions of dollars. Much of this capital cost is attributed to the specialised equipment and tooling required

to process the semiconductors. Since actual equipment cost is a fixed capital expenditure (generally a one-off purchase), and cannot be recovered, a substitute is to use the equipment amortisation cost instead. This amortisation cost can be offset by running tools and equipment at a very high level of utilisation. This means that minimising fab costs can be done by maximising the utilisation of factory equipment. Unfortunately though, queue time, inventory and work in process (WIP) levels are negatively impacted by this policy, and cycle time increases. This means that the average time that material spends queueing (as a ratio of the time spent processing) increases rapidly, and the material workflow lines become more congested. This is a very unfavourable situation and can cause important capital to be tied up in inventory, resulting in longer lead times and reduced cash flow for the parent company.

These two related factors are described by operational characteristic curves (or operating curves for short). Sematech, an organisation representing a consortia of semiconductor manufacturers, produced the Measurement and Improvement of Manufacturing Capacity (MIMAC) technical report in 1995 which discussed the key performance indicators (KPIs) of a semiconductor fab. Of all the consortium members consulted in the report, most stated that the key performance analysis should be based on operating curves (Fowler and Robinson, 1995). Similarly Ignizio (2009) stated that the operating curve is the single most important metric for a factory manager. In light of this, this thesis focuses on this particular metric, which can be related back to the economic drivers of the semiconductor industry. The following sections explain the fundamentals of operating curves and the advantages of using them as a factory metric. There is also a discussion on the factors that influence the shape of the curves, and hence, have an impact on overall fab performance.

2.2.1 Queueing systems

In manufacturing systems, the entities that reside in the system compete for finite resources. For example, lots in a wafer fab must compete and wait (queue) for tools,

operators and AMHSs. It is this competition that forms a series of queueing systems around the fab. In queueing theory, the items that need the resource are known as customers and the resources that provide a service to the customers are known as servers. Queueing systems are often described using Kendall notation (see pg. xv).

General queueing approximations

One of the benefits of analysing systems using queueing theory is that much work has been done to derive the estimators of the long run performance measures for many of the less complex idealised systems. For example, the cycle time (the average time from when a lot or job is released into a system to when it exits) for an M/M/m queue, that is, a queue with exponentially distributed arrival and service patterns is given by Eq.(2.1) (see Hopp and Spearman (2001) for derivation) as follows,

$$CT_{M/M/m} = t_e \left(1 + \frac{u\sqrt{2(m+1)-1}}{m(1-u)} \right), \quad (2.1)$$

where m is the number of tools in the toolset, u is the utilisation of the toolset (the fraction of time the toolset is not idle for lack of WIP) and t_e , the mean effective process time, refers to the times ‘seen’ by the lots, which incorporates time spent in repair, setup changes, loading times and any other production detractors. Utilisation can also be defined as,

$$u = \frac{t_e}{t_a m}, \quad (2.2)$$

where t_a is the mean inter-arrival times of lots to the toolset. For a single server system ($m = 1$), Eq.(2.1) reduces to,

$$CT_{M/M/1} = t_e \left(1 + \frac{u}{1-u} \right) = \frac{t_e}{1-u} \quad (2.3)$$

The M/M/m queue is a uniquely special case due to the *memoryless* property of the exponential distribution. The memoryless property means that the time of the next event

is independent of the time already spent waiting for the event.

In a semiconductor manufacturing environment, the exponential distribution can be a good approximation for the arrival pattern to a tool if there are multiple independent upstream lot sources feeding the tool. Again, this stems back to the memoryless property of the exponential distribution. Unfortunately however, the exponential distribution is not a good model for tool processing, given that its lower bound is zero and it is likely that very small process times can be sampled from the distribution. Tool processing is somewhat better served by using a ‘time to perform a task’ distribution such as the lognormal distribution (Hopp et al., 2002; Law, 2008). A benefit of using the lognormal distribution is that its shape is more accommodating to a process with a stochastic pattern that has a left sided hard floor, below which, samples are highly unlikely or impossible.

In order to use the lognormal or any other general queueing distribution, a G/G/m queue is employed, whereby the arrival and process pattern are represented by general distributions. Equation (2.4), derived by Kingman (1966) and explored further by many queueing theorists, in particular Whitt (1983, 1993), is often used to estimate the cycle time performance of a queueing system with arrival and service patterns from a general distribution.

$$CT_{G/G/m} = t_e \left(1 + \left(\frac{c_a^2 + c_e^2}{2} \right) \frac{u\sqrt{2(m+1)-1}}{m(1-u)} \right) \quad (2.4)$$

The difference between Eq.(2.4) and Eq.(2.1) is the addition of the squared coefficient of variability terms for the mean inter-arrival times and mean effective process times, namely c_a^2 and c_e^2 .

Variability is traditionally measured using the standard deviation divided by the mean, known as the coefficient of variation c . If t is the mean (t is used because the random variable of interest here is time) and σ is the standard deviation then $c = \frac{\sigma}{t}$. It is sometimes more convenient to use the relative measure, the squared coefficient of variation c^2 , where $c^2 = \frac{\sigma^2}{t^2}$.

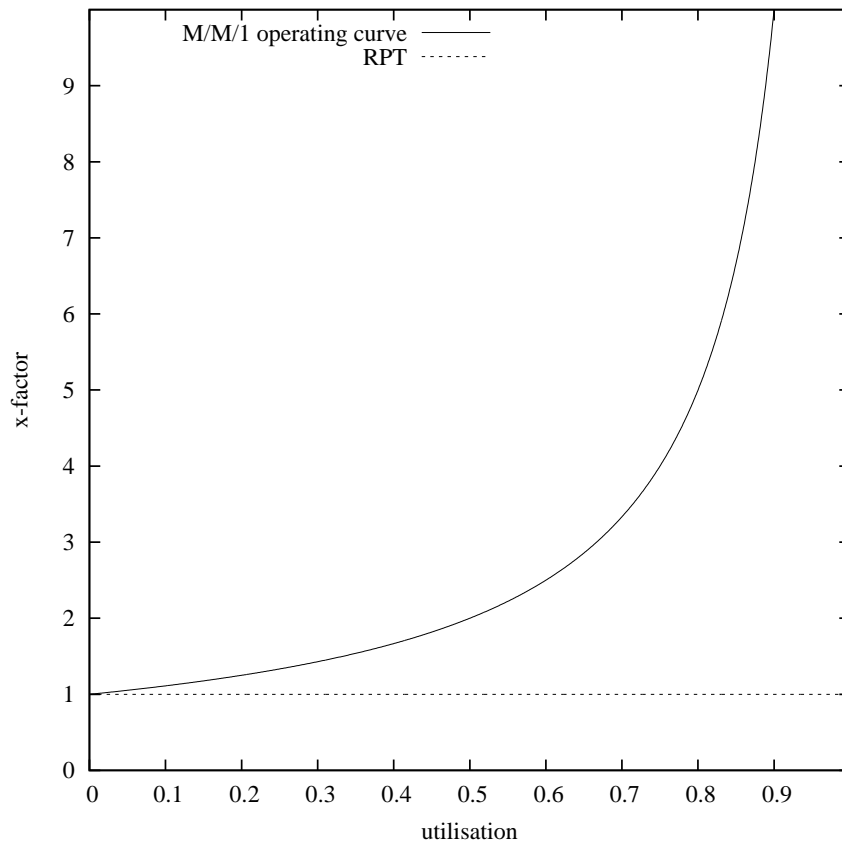


Figure 2.3: Operating curve for an M/M/1 queueing system showing the relationship between x-factor and utilisation.

It can also be useful, to show cycle time as a proportion of the raw process time (RPT) (t_0), where RPT refers to the average time for a single job or part to traverse an empty system. This ratio of cycle time to raw process time is known as the *x-factor* (or normalised cycle time) and gives an indication of the ratio of the queueing time to process times, as in Eq.(2.5). Plotting the relationship between x-factor and utilisation gives an operating curve, an example of which is shown in Fig. 2.3 for an M/M/1 queue.

$$x_{\text{basic}} = \frac{\text{CT}}{\text{RPT}} \quad (2.5)$$

It can also be seen from Fig. 2.3, that as utilisation increases, the corresponding cycle time increases exponentially. As utilisation approaches 100%, cycle time goes to infinity. Hopp and Spearman (2001) classified *Law 8* in *Factory Physics*; “If a system increases utilisation without making any other changes, average cycle times will increase in a highly

non-linear fashion”. The highly non-linear fashion referred to in *Law 8* causes cycle times and WIP to “blow up”. Meaning, not only does cycle time increase as utilisation increases, but it does so at a very fast rate which causes the system to become highly unstable.

2.2.2 The advantages of using operating curves

If a factory’s operating curve can be successfully mapped, then management can gain a better understanding of how their factory behaves. Olhager and Persson (2008) pointed out that fundamental relationships between variables displayed on an operating curve can aid in increasing the knowledge about a system and can be used as a decision support tool. Many other authors including Aurand and Miller (1997); Fayed and Dunnigan (2007); Fowler et al. (1997); Li et al. (2005); Potti and Whitaker (2003); Sattler (1996); Veeger et al. (2008) discuss the simplicity and intrinsic benefits of using operating curves as a key factory metric.

One of the most useful characteristics of operating curves is that they can be used to monitor systems and sub-systems from factory level down to single process level. Meaning, it is possible to have an operating curve for the full fab, a functional area, a machine group, a single machine or even a process through a single machine.

The shape of the curve defines the efficiency of the system. The contours on the curve are a snapshot of the system’s response across a full loading profile and an indication of the performance of the system across its bounds of operation. Any particularly steep inclines or pronounced inflection regions on the curve can help to identify economic thresholds.

For example, Fig. 2.4 shows the typical region where one would find the curve inflection for an M/M/1 queueing system according to Eq.(2.1). An optimum level of operation exists just before this inflection region, where the system can be loaded without a significantly disproportionate negative impact to cycle time. However, after this region, any increase in loading results in a steep exponential rise in cycle time.

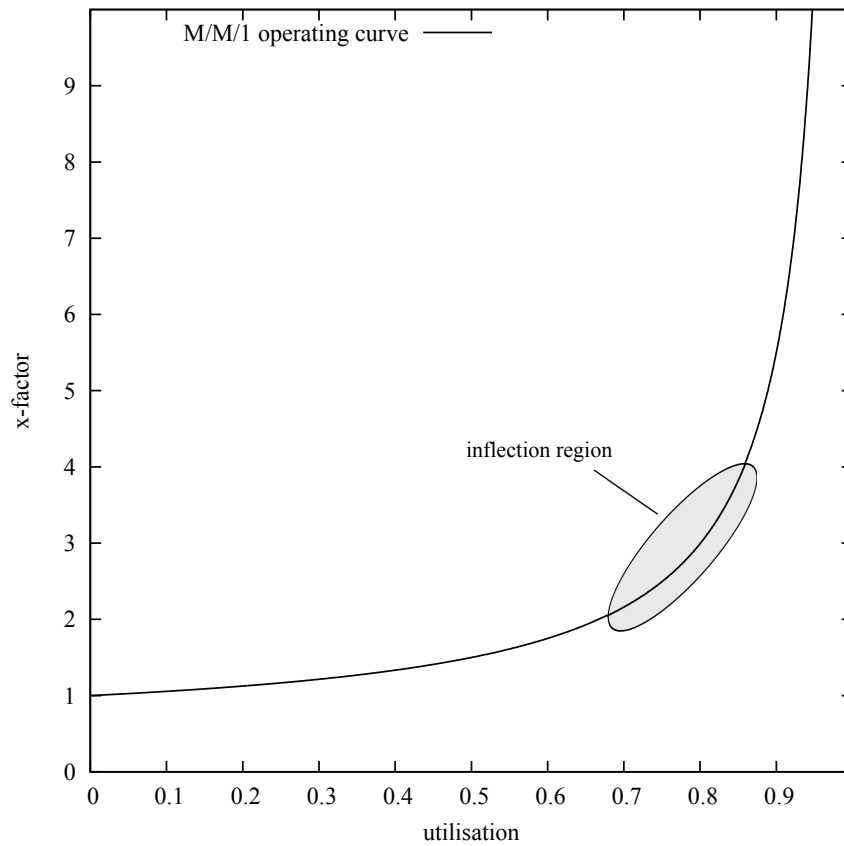


Figure 2.4: Typical inflection region on an M/M/1 queue operating curve.

2.2.3 Factors that impact fab performance

Operating curves are key descriptors for a factory. Any significant changes on the factory floor are likely to cause some shift in the factory curve. Based on interviews with leading experts, Fayed and Dunnigan (2007) listed 19 key factors that have a significant impact on operating curves. They categorised them under four main headings; fab configuration, fab loading, flexibility & variability, and operations. Controlling these factors is key to a ‘healthy’ factory with an operational characteristic that is low with a right-shifted more defined inflection region. The following is a list of Fayed and Dunnigan’s 19 factors.

A. Fab configuration

1. Fab size
2. Fab layout
3. Equipment standardisation

B. Fab loading

4. Capacity utilisation
5. Product mix
6. Product type
7. Starts configuration and steady loading
8. Product control and holding
9. Number of constraints

C. Flexibility and variability

10. Recipe flexibility
11. Equipment breakdown variability
12. WIP and inter-arrival variance

D. Operations

13. Hold lots
14. Hot lots
15. Operator availability/level of automation
16. Scheduling and dispatching
17. Defect inspection sampling
18. Operations
19. Overall equipment effectiveness (OEE) management

One of the benefits of having an accurate operating curve is that it allows management to adjust capacity based on a target cycle time. For example, market forces generally dictate a minimum lead time for products. In order for a fab to remain competitive it must keep its cycle time within range of the industry lead times. Using an operating curve, management can estimate the likely utilisation required to maintain this cycle time. If such a cycle time is impossible to achieve without high utilisation then additional capacity can be added to the system.

2.2.4 Fab utilisation and bottlenecks

At any given time a fab should be operating at some point along its operating curve. As long as the system is relatively stable then increasing or decreasing loading should move this point to the left or right in a pattern described by the factory's curve. In order to

generate an operating curve it is necessary to map this line and determine the current operating point. Two metrics are required to locate the current operating point, the factory cycle time and utilisation level. The cycle time metric is usually easy to find as most fabs will have some system in place for recording this. However, estimation of the factory utilisation is more complex and is not always a deterministic value (Butler and Matthews, 2001).

Bottlenecks

A local indicator such as the utilisation of a toolset may be more convenient to use as a value for overall fab utilisation (Aurand and Miller, 1997; Martin, 1996). The toolset that most represents the factory is generally the bottleneck toolset.

The theory that a factory's capacity is effectively equal to the capacity of its bottleneck toolset is one that is used by the theory of constraints (TOC) (Goldratt, 1990, 1992). TOC attempts to focus on the control of WIP through the bottleneck or 'constraint' toolset by ensuring that it is never starved of WIP and its downstream toolsets have sufficient capacity, not to cause a blockage to the constraint. This manufacturing technique attempts to match the factory flow speed and throughput to that of the constraint toolset throughput.

Identifying the bottleneck is a non-trivial task, particularly in a complex semiconductor fab. Some fabs experience floating bottlenecks, whereby a number of near constraint tools can be considered the bottleneck at any given time (Koo et al., 2005). This is particularly prevalent in fabs that experience high WIP fluctuation and WIP clusters. The more regulated the flow of product through the factory the more consistent the location of the constraint toolset.

There are different methods of identifying the bottleneck; the most common one is generally to point to the toolset with the highest utilisation or the toolset with the least capacity (Hopp and Spearman, 2001).

Another issue with bottleneck identification is the assumption that a toolset or a

tool is always the bottleneck. This is not always true, sometimes the bottleneck can be some other fab entity such as an AMHS. In light of this, Li et al. (2009) proposed a system of identifying bottleneck areas based on the probability of starvation and blockage. Their analytical metric identified the line sections that had the largest impact on total throughput. Overall however, it is a rarity that this occurs, and in the main, bottlenecks are generally toolsets or workstations (Koo et al., 2005).

Assuming that the factory *is* represented by a bottleneck toolset, the assumption then is that the operating curve of the bottleneck toolset has similar characteristics as that of the whole fab. These assumptions may be more acceptable when the bottleneck is clearly identifiable and the near constraint toolsets have significantly lesser impact on the fab. However, if there are a number of toolsets that are tied for selection as the bottleneck and there is no clear ‘winner’, or the bottleneck toolset is not much ‘worse’ than the other toolsets in the fab, then it is generally not a valid assumption to use one singular toolset as the bottleneck.

Tool utilisation

If an appropriate bottleneck can be found, further complications arise in the calculation of utilisation, of which there are two different approaches; loading based and output based Lopez et al. (2005). The loading based metric is calculated from the equipment runtime percentage as in Eq.(2.6).

$$u_{\text{load based}} = \frac{\text{Equipment Running Time}}{\text{Total Equipment Uptime}} \quad (2.6)$$

The advantage of using a load based metric is that both the equipment running time and uptime information are usually easy to retrieve, and the calculations are relatively simple. However, the disadvantage of this method is that the metric can be unrealistic when dealing with more complex tools. “Many equipment utilisation systems are set up to monitor and track utilisation relative to a machine’s expected available capacity. In other words, utilisation of available capacity is often tracked, while utilisation of a machine’s

theoretical maximum capacity is usually not tracked” (Aurand and Miller, 1997). Load based utilisation calculations do not recognise the difference between a tool running and a tool running optimally. For example, a batch tool running one lot as opposed to its maximum capacity of a batch of five is reported as 100% utilised for that period, even though the actual output was only a fifth of what was theoretically possible.

$$u_{\text{output based}} = \frac{\text{Actual Output}}{\text{Theoretical Output}} \quad (2.7)$$

Alternatively, output based utilisation is a ratio of the actual output to the theoretical output (Eq.(2.7)). This addresses the issue of a tool running sub-optimally. However, the difficulty when using an output based metric is in obtaining an accurate estimation of theoretical output. For example, if a tool is only capable of processing lots in a particular sequence, then its maximum theoretical output is largely dependent on having sufficient WIP and correct product mix to achieve that maximum output. If the optimal mix is unavailable then the theoretical maximum output value needs to be re-evaluated for such a scenario.

2.3 Modelling the Fab and its Operating Curve

As shown, operating curves can be used to provide a snapshot of fab performance and how it reacts to changes in loading levels. Operating curves are essentially a model of the factory, and like all models, they should be constructed using a correct methodological approach. The following sections describe the process of modelling complex systems with a focus on the two most appropriate techniques for generating operating curves; analytical approximation and discrete event simulations.

No substantial part of the universe is so simple that it can be grasped and controlled without abstraction. Abstraction consists in replacing the part of the universe under consideration by a model of similar but simpler structure. Models . . . are thus a central necessity of scientific procedure.

Rosenblueth and Wiener (1945)

Before one even contemplates analysing a system, the mind immediately attempts to offer some interpretation of the system. Indeed, immediately one can probably get a grasp of the most dominant mechanisms that control the variables of the system. On further inspection it may be possible to gain a very good insight into the system but it will never be possible to capture 100% of the system (Law, 2008).

For engineers, scientists, factory managers and most system owners there is a need to bring order to chaos. The role of the engineer and scientist is to attempt to classify a system and control it, to make it more efficient. To do this, it is necessary to be able to experiment with the configuration of the system and investigate how the system will operate under alternative policies and configurations. Experimenting with the configuration can take place in two forms, experiments with the actual live system or experiments with a model of the system. Experiments with the actual system deliver a great deal of insight, however, these tests are not always feasible. It may not be possible to do actual tests because the system may not even exist, or the proposed changes may not be economically or logistically possible without some confirmation of their probable success.

Modelling the system is the cheaper alternative. The key disadvantage of modelling the system is that no model can ever be 100% accurate. As a result, some form of trade-off must exist between the cost of providing a modelling solution and the precision of the model. The choice of the type of model is as important as the scale and scope of the model. System modelling falls into two categories, physical models and mathematical models, displayed in Fig. 2.5. Physical or *scale* models are far less popular than building some mathematical interpretation of the system.

The type of mathematical model is then further defined by the type of solution used to solve the model. An analytical model is generally considered as one where the variables have distinct quantities and the model is solved by deductive reasoning to find an exact solution. Analytical models may be simple equations with only a small number of parameters, such as Little's Law. More complex analytical solutions can involve a series

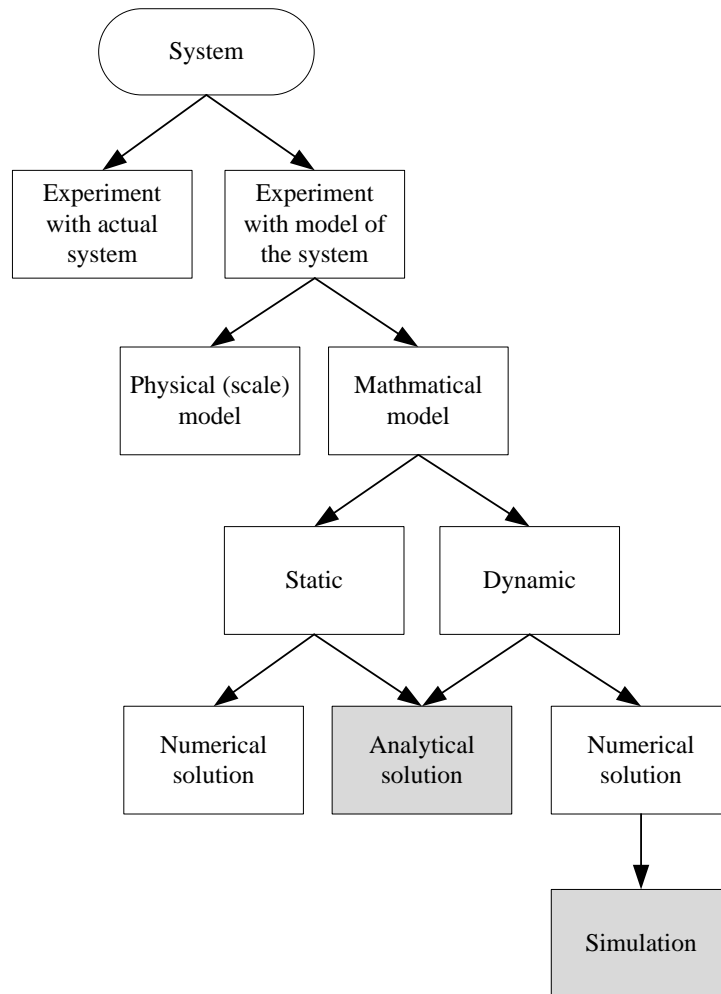


Figure 2.5: System modelling methods, *based on* (Gordon, 1977; Law and Kelton, 1997).

of mathematical solutions that must be solved to acquire unique values.

Alternatively, the system may be of such complexity that it becomes easier to use simulation. A simulation model is one whereby the mathematical model is tested with a series of inputs and solved using numerical methods. Some authors such as Banks and Gibson (1997a); Banks et al. (2004); Law and Kelton (1997); Pidd (1992) state that simulation should only be used as a last resort and should never be used when a simpler, more tractable, analytical model is available. However, Rubinstein and Melamed (1998) argued that the advances in modern simulation methodologies and user friendly software have made it much more accessible and feasible.

The focus of this thesis is on generating operating curves using both static analytical solutions and dynamic, statistical, discrete event simulations. The following sections examine both of these methods and offer a critical evaluation of their ability to model a semiconductor fab and ultimately generate an accurate operating curve.

2.4 Generating Operating Curves using Analytical Modelling Methods

Analytical modelling involves the deployment of a mathematical formula with a closed form solution. The term *analytical methods* is used to mark a distinct difference between closed form solutions and numerical methods (simulation modelling being a very common numerical method). More specifically, when using the term ‘analytical models’ to describe manufacturing systems that can be expressed as queueing networks, the focus changes to queueing modelling.

2.4.1 Modelling semiconductor fabs using analytical models

The most accurate method of generating an operating curve is to run actual experiments on the factory floor and test the cycle time response over a range of loading levels. This would be extremely time consuming, expensive and is generally an unrealistic proposition. Instead, an alternative is to test the fab at the current operating point and infer the cycle time at other loading levels using analytical approximations. There are several techniques for generating curves that use analytical methods, but the steps are typically the same:

1. Plot the zero utilisation RPT line at an x-factor of 1 (as in Fig. 2.6),
2. Using actual cycle time data and the RPT construct the current x-factor horizontal line,
3. Select an analytical approximation and construct the curve,
4. The intersection of the x-factor line and the operating curve give an approximation of the current factory operating point.

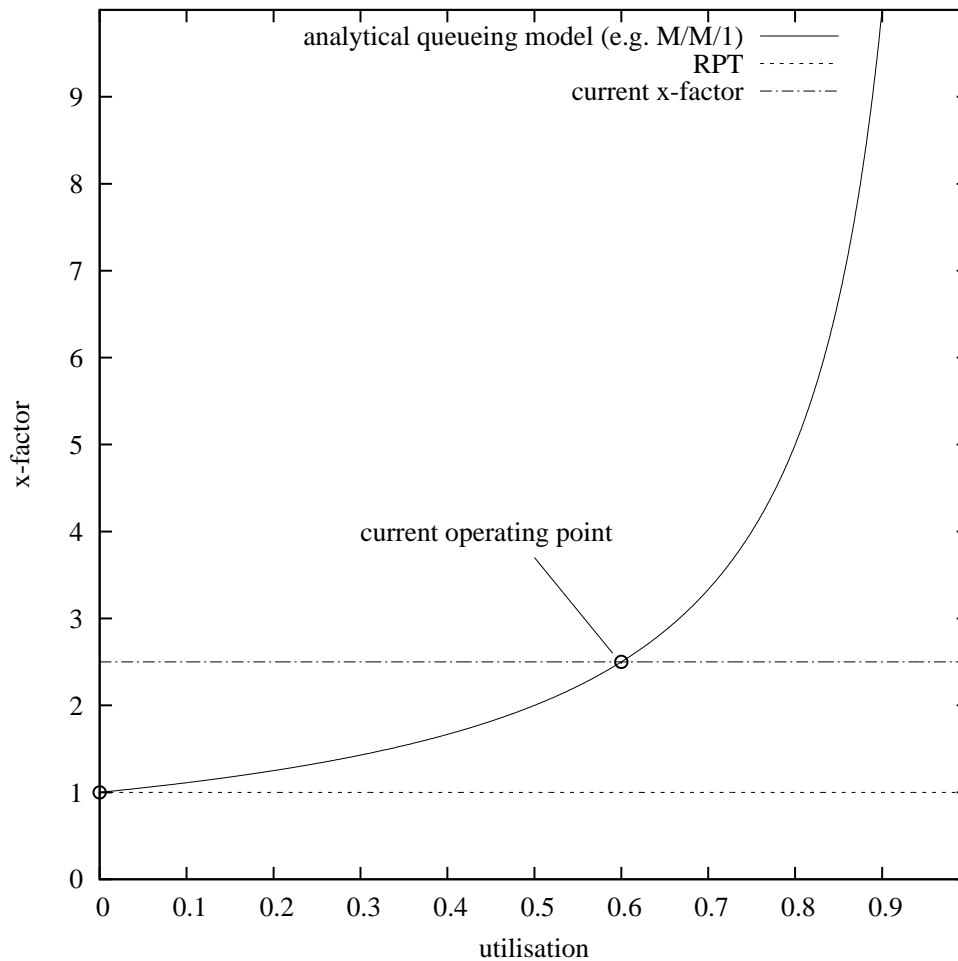


Figure 2.6: Locating the current operating point on an operating curve generated by an analytical queueing model.

Step 3 is the most important step. Selection of the model is critical to the accuracy of the operating curve. The process is typically based on some theoretical premise of how the fab operates. It is possible to use the M/M/m and G/G/m queueing approximations given by Eqs.(2.1)-(2.4), however, it is more often the case that these queueing systems are expanded or amended to suit the system under observation.

Complete x-factor contribution measure

Common practice involves manipulation of the common queueing approximation for the system under observation. One such version, given by Delp et al. (2006), is the complete x-factor contribution (CXFC). The CXFC is derived from Martin (1996) and was further developed by introducing an *unavailability coefficient* for each machine group to account

for downtime by Delp et al. (2003); Delp (2004); Delp et al. (2005). It attempts to get a measure of the cycle time ‘contribution’ from each tool group as a proportion of the total x-factor of the system. The x-factor contribution κ for a toolset j is a function of the normal x-factor of the G/G/m queueing component, the raw process time for the toolset t_{0j} , the system raw process time t_0 and the downtime unavailability coefficient V . V can be thought of as the probability that an incoming lot will find all tools within the toolset offline at the same time as a result of their individual downtime instances coinciding.

$$\kappa_j = (1 + V_j) \frac{t_{0j}}{t_0} \left(1 + \left(\frac{c_{a_j}^2 + c_{e_j}^2}{2} \right) \frac{u_j^{\sqrt{2(m_j+1)}-1}}{m_j (1 - u_j)} \right), \quad (2.8)$$

where the unavailability coefficient V_j is given by,

$$V_j = \frac{(1 - A_j)^{m_j}}{m_j + 1} \cdot \frac{t_{r_j}}{t_{0j}}, \quad (2.9)$$

and the availability A is a function of the mean time to repair (MTTR) t_r and the mean time before failure (MTBF) t_f for the toolset, as in Eq.(2.10), assuming they are both exponentially distributed.

$$A = \frac{t_f}{t_f + t_r} \quad (2.10)$$

Although there is no evidence presented by Delp et al. that the CXFC is better than using the x-factor alone, the inclusion of the unavailability coefficient suggests that it should be better, and at the very least more inclusive. Also, the fact that it provides an alternative method to identify ‘constraints’ is more comprehensive.

The V component used by Delp et al. is based on a downtime factor used by Martin (1996), which assumes that the MTBF and the MTTR of any machine in the toolset are both exponentially distributed. For the MTBF this is a ‘safe’ assumption, particularly for advanced machinery, like that in the semiconductor industry, where the sources of failure are numerous. Such plentiful failure sources benefit from the exponential distribution’s

memoryless property, similar to how arrival patterns are well described by the memoryless arrival events simulated by multiple upstream sources. MTTR is better served by a *time to perform a task* distribution such as the lognormal or the Johnson family of distributions (see Appendix F for discussion of the benefits of using the Johnson distribution). However, the elegance of the solution and the simplicity of the derivation of the V_j factor is based on the simplicity of implementation of the exponential distribution and therefore, any attempt to incorporate a general distribution would make the CXFC measure a less tractable solution.

Queue approximation for *idle with WIP* problem

Simple closed form queueing approximations, such as those given by Eqs.(2.1)-(2.4), assume that once a server becomes idle and there is WIP available, then a lot gets loaded onto the server or tool instantaneously. However, in a semiconductor manufacturing environment, there are some situations when this is not the case. An example of this is when an operator is not available or some other piece of equipment that is required by the tool (e.g., a reticule at a photolithography tool) is not available.

Such occurrences require an additional delay factor to be incorporated into the queueing approximation. Morrison and Martin (2007) proposed a solution based on the assumption that a lot experiences a preproduction random delay (when the tool is idle) that follows a general distribution pattern described by its mean t_I and standard deviation σ_I . The mean effective process time t_e is then given as follows,

$$t_e = \frac{t_0 + t_I}{A}, \quad (2.11)$$

where A is the general tool availability, (e.g., $A = 1$ for a tool that is always online). Based on a G/G/m queue with a mean t_r and squared coefficient of variability c_r^2 of repair time, substituting Eq.(2.11) into Eq.(2.4) yields,

$$\text{CT}_{G/G/m, I} = \frac{t_0 + t_I}{A} \left(1 + \left(\frac{c_a^2 + c_{e,I}^2}{2} \right) \frac{u\sqrt{2(m+1)-1}}{m(1-u)} \right), \quad (2.12)$$

where,

$$c_{e,I}^2 = \frac{\sigma_e^2 + \sigma_I^2}{(t_0 + t_I)^2} + (1 + c_r^2) A(1-A) \frac{t_r}{t_0 + t_I} \quad (2.13)$$

Equation (2.12) can then be used to estimate the cycle time for the queue with a random reproduction delay. However, the approximation assumes a single reproduction delay pattern is in existence, multiple independent delay patterns may require further expansion of the approximation.

2.4.2 Benchmarking fabs using operating curves

The dependence on the accuracy of an operating curve can be minimised by using operating curves to benchmark fabs. Instead of using operating curves to predict performance under varying loads, they are used to monitor the fab and to benchmark against past performance. Therefore, as long as a consistent methodology is applied, an operating curve monitored over time can be used to at least examine if any gains or losses have been made in the efficiency of the fab.

Benchmarking using the P-K formula.

One such practical implementation of generating operating curves based on a queueing approximation was performed by Aurand and Miller (1997). They used an M/G/1 queueing system (sometimes referred to as the Pollaczek and Khinchin (P-K) formula) with exponentially distributed arrivals, generally distributed process times and a single server to represent a black box model of an IBM fab. Letting $c_a^2 = 1$ and $m = 1$ and using the raw process time t_0 instead of mean effective process time t_e , Eq.(2.4) reduces to,

$$\text{CT}_{M/G/1} = t_0 \left(1 + \left(\frac{1 + c_e^2}{2} \right) \frac{u}{(1-u)} \right), \quad (2.14)$$

The u value was based on the capacity utilisation of the bottleneck tool with the assumption that it was representative of overall fab capacity (see Section 2.2.4 for discussion on the merits of this assumption). The bottleneck tool was selected as the tool with the smallest maximum theoretical output if there were no production detractors affecting that tool (as in Eq.(2.7)). The raw process time t_0 was measured using the weighted average of the sum of the process times for each product type. Non value added operations, such as transport and inspection were not included. The cycle time was found from historical fab data averages.

The remaining unknown variability factor c_e^2 , was then calculated from Eq.(2.14) using the known values for u , CT and t_0 . Using this c_e^2 value allowed them to plot an operating curve and repeating this process every few months allowed them to benchmark the factory.

Key to this method was the strict methodology employed. As long as the method was repeated, the curves were useful for benchmarking. However, it was not possible to benchmark against other factories, a disadvantage stressed by Aurand and Miller, due to the broad assumptions made.

Another issue with this method is fixing of the unknown process variability factor c_e^2 . The method assumes that this does not change over the period of analysis and all future curves take this fixed value. Furthermore, one would question the assumption of $c_a^2 = 1$. If the model is a ‘black box’ of the whole fab, then it is likely that arrivals to the fab (the *starts rules*) are not exponentially distributed, but perhaps, more constant as is usual in a real fab. Nevertheless, as a benchmarking exercise, the resulting operating curves at least indicated to management whether or not improvement gains had been made in the fab.

The O-L graph method

Li et al. (2007) used a similar approach to construct their operating curve (or O-L graph) for a toolset based on Little’s Law. Again, they stated that capturing the system vari-

ability was too complex and they attempted to isolate the variability factor like Aurand and Miller (1997), but instead used a G/G/m queuing model for the system as opposed to using the M/G/1 model. Their operating curve was based on the assumption that over a relatively short period of time the variability in the system was constant. This meant that it was possible to capture operating points at different load levels at different times and assume that they were all on the single operating curve. The formula they used is given in Eq.(2.15), which is a rearrangement of Eq.(2.4), where $c^2 = \frac{c_a^2 + c_e^2}{2}$.

$$c^2 = \frac{m(1-u)}{u\sqrt{2(m+1)-1}} \frac{CT - t_e}{t_e} \quad (2.15)$$

They also went a step further than Aurand and Miller by including setups and downtime in their calculation of t_e . Again, however, the applicability of their method for anything other than benchmarking is unclear and the validity of assuming a fixed variability factor may be unrealistic if conditions at the toolset change rapidly.

Fab performance function based on the G/G/1 queuing model

Primarily tasked with benchmarking a semiconductor fab by capturing the trade-off in cycle time and utilisation, de Ron and Rooda (2005) used a G/G/1 queuing approximation. They assumed that their fab performance metric P could be expressed as a quotient of throughput-cycle time ratio for the actual system and that of a reference system. If the reference system is the theoretical best performance of the manufacturing system, then the best possible throughput is that of the maximum theoretical bottleneck throughput TH_0 and the best possible cycle time is the raw process time t_0 . Using the G/G/1 queue as a model for the current configuration, P was given as,

$$P = \frac{\left(\frac{TH}{CT}\right)}{\left(\frac{TH_{ref}}{CT_{ref}}\right)} = \frac{TH}{TH_0} \cdot \frac{t_0}{CT_{G/G/1}}, \quad (2.16)$$

All of these examples of fab benchmarking using operating curves are based on the assumption that the fab or a fab toolset can be captured as a ‘black box’ by a queuing

model. However, none appear to question the validity of using queueing theory in a semiconductor fab, which has some inherent fundamental shortcomings.

2.4.3 Implications of the fundamental assumptions associated with queueing theory

These analytical models offer a good base for understanding the nature and behaviour of a fab and as seen, can be useful as a benchmarking tool, but there are other more fundamental issues with the application of analytical queueing models to complex semiconductor manufacturing that question the validity of their applicability in such an environment.

Non-identical toolsets and variable toolset number

One of the key issues is the existence of non-identical toolsets which some authors also refer to as the ‘non-parallelism’ of toolsets (Shanthikumar et al., 2007). This thesis shall refer to it only as non-identical toolsets to distinguish it from the term ‘processing parallelism’ of tools which refers to certain complex tools that can process more than one item simultaneously.

Queueing models assume that all tools within a toolset have an identical service pattern, and that a lot can select any of the tools if they are idle. However, in semiconductor manufacturing, tool dedication and equipment standardisation affect the choice of tools within a toolset that a lot can select from. This dedication system was brought in to increase the yield of good wafers, but generally has a negative impact on queueing and cycle time, given that the effective capacity that a lot *sees* at a toolset is reduced, (see Section 2.1.3 for a more detailed discussion). For example, assuming that a lot enters a toolset and its choice is unrestricted, then the capacity of the toolset is equal to the number of available tools. If however, a lot is dedicated to a particular tool, the incoming lot only *sees* one tool, meaning that the *capacity* is effectively reduced to one tool.

A similar situation occurs because of setups and equipment standardisation. Some-

times tools within a group are only qualified to process certain layers, or have model differences with other tools. This causes complications when trying to estimate the parallel capacity during the application of queueing models. Miltenburg et al. (2002) encountered this when they applied queueing network models to a number of semiconductor facilities. In one particular facility, four out of 20 stations within the fab could not be classified in the queueing models because the tools within their respective toolgroups differed so much.

Similarly, Juang and Huang (2000) stated that the nominal tool number m of a toolgroup could not be used in queueing formulas because of “heavy overlapping” of toolset boundaries. Instead, they suggested using a modified variable known as the effective tool number. If $W_{q,g}$ is the mean waiting (queueing) time for a lot at toolset g , and $f(c_a^2, c_e^2, t_0, \lambda, c)$ is a function of the toolset, where λ is the lot arrival rate and b is the toolset vector $b = (b_1, b_2, \dots, b_G)$. Then the effective tool number m^* was given by,

$$m^* = \min_b \left(\sum_g \left| f(c_a^2, c_e^2, t_0, \lambda, b) - \frac{\sum_{i=0}^n w_{q,g,i}}{n} \right| \right), \quad (2.17)$$

with n observed waiting times $w_{q,i}$ for each tool group. The $f(c_a^2, c_e^2, t_0, \lambda, c)$ function is derived by finding the values c_a^2 , c_e^2 , t_0 and λ from real system data and substituting them into a closed form queueing approximation such as that of Eqs.(2.1) to (2.4). The waiting time approximation is solved for m^* against actual waiting times. While this method is quite practical it requires sample data from the system and sufficient number of observations n .

Juang and Huang also noted that an important insight could be gained from using this method for ranking toolgroups by the ratio of their effective tool number against the actual nominal tool number. This could help identify lowly ranked toolsets where more standardisation might be required. Also, if the ranking was very low (a very large difference between nominal and effective tool number) it could be easier to reclassify the

toolset into individual tools and assume a number of single dedicated tool configurations.

As mentioned above it is not always possible to classify a toolset by a single integer number of tools m as required for most closed form queueing approximations. It is more likely that the number of tools in a toolset is a dynamic real value that changes depending on the type of WIP and the conditions of rework and dedication.

In light of this, Sattler (1996) proposed placing m on a range $m^* \in [1, m]$, where $m^* = 1$ implies that there is 100% dedication and that the lot can only select one particular tool from the toolset. If $m^* = m$ then there is no dedication and lots are ‘free’ to choose any tool within the toolset. Their method involved use of a heavy traffic cycle time approximation for a G/G/m queue given by Gross and Harris (2003) as follows,

$$CT_{G/G/m \text{ (heavy traffic)}} = t_e \left(1 + \left(\frac{u}{1-u} \right) \frac{m^{*2} \sigma_a^2 + \sigma_e^2}{2m^* t_e^2} \right), \quad (2.18)$$

Letting $k = \frac{m^{*2} \sigma_a^2 + \sigma_e^2}{2m^* t_e^2}$ and solving Eq.(2.18) for k , meant that they could fix this variable and use it for future operating curves.

Similar to Aurand and Miller (1997) and Li et al. (2007), they assumed that the variability (or standard deviation of variability in this case) measured at a base or reference point does not change when measured at another point in the future.

Besides the difficulty in estimating the number of tools within a toolset, there is also an issue if different process patterns are observed by tools within the same toolgroup. However, all of the queueing approximations discussed in this chapter have assumed that there is one single stochastic mechanism driving the processing time component of the approximations.

Jacobs et al. (2001, 2003) addressed the issue of machines that had different processing patterns (calling them ‘unequal’ tools) by implementing an effective process time (EPT) algorithm that could generate a value for t_e and c_e for individual non-identical tools within a toolgroup based on a list of arrival and departure events of lots to each tool. However, it did not estimate a value for m that could be used in a queueing approximation.

Arrival-service-WIP independence assumption

Other complications when implementing queueing formulae to semiconductor manufacturing arise because of the fundamental assumption of independence of arrival and service patterns (Jacobs, 1980; Shanthikumar et al., 2007). means that regardless of the arrival distribution, the process distribution does not change. However, in a real fab situation, arrival and service patterns are inherently linked and cannot be assumed to be independent, particularly in a fab where management implement a range of WIP control strategies.

For example, if engineers believe that a particular section of the fab or toolset will be impacted by a WIP cluster then it is sometimes possible to smooth out this cluster by employing certain techniques and strategies. Examples of such strategies include; delaying PM schedules, altering the batch sizes, or minimising setups by processing as much as possible of a particular product type before having to perform a changeover. In such situations, the processing pattern is being influenced by arrivals and it cannot be said that the service and arrival patterns are independent.

One potential solution, proposed by Akhavan-Tabatabaei et al. (2009), is to apply Little's law to bucket intervals of historical data to determine the cycle time during various WIP scenarios. However, this method seems to be somewhat divorced from queueing theory approximation, and seems more in the category of numerical approximation. Progress in this area is questionable and (Shanthikumar et al., 2007) stated that there appears to be no research that has successfully addressed the issue of dependency between arrival and service pattern in semiconductor manufacturing.

2.4.4 Queueing networks

The aforementioned issues show the fundamental shortcomings of queueing theory when being applied to semiconductor toolsets. An even more difficult proposition is to model the factory as a queueing network. In a queueing network, the output from one queueing

node supplies the input for another. Although there is much literature (e.g., Connors et al. (1996); Hopp et al. (2002); Juang and Huang (2000); Whitt (1983)) on the subject of queueing network models, according to Kotcher and Lumileds (2011) they were unaware of any fab that was using a queueing network model on a day-to-day basis.

2.4.5 Advanced queueing approximations

In an effort to deal with the complexities of semiconductor manufacturing it may be necessary to lose some tractability and use more advanced queueing theory analysis that cannot be approximated by the convenient closed form solutions of Eqs.(2.1)-(2.4). For example, Lee and Kim (2005) described a fab as a multi-product production system in a varying environment and suggested a queueing model for a system with;

- multiple types of products,
- multiple machine conditions,
- decisions on the acceptance of orders based on machine conditions,
- process times that are dependent on machine conditions.

Combining this type of system along with some of the practical extensions to account for fab phenomena such as rework, re-entrancy, batching, cascading, non-identical tools and independence between arrival and service patterns, would offer more accurate approximations of performance measures and deliver more accurate operating curves. The problem is that many of the approximations for dealing with semiconductor fab phenomena are designed for particular instances or scenarios and few offer an overall generic model. This is because the more aspects that are included in the model, the more complex the analytical models must become, the more complicated solving them is and the less tractable and difficult they are to implement. Furthermore, any attempts to characterise a fab using complex queueing models requires experienced queueing theory practitioners. Using such complex analytical models may begin to move away from their fundamental use; that engineers and management can gain a better understanding of how their fab

operates.

In general, analytical methods are best applied to a simple more idealised system and their advantages lie in the tractable solutions offered. However, if examining the true underlying operational characteristic of a complex system, such as a semiconductor fab, their deployment is cumbersome and difficult. Many of the methods previously discussed involve idealising and fitting the factory to a curve and fixing some variables to track any changes in the fab. Such methods are flawed because they attempt to idealise the fab for the purposes of fitting a model, as opposed to testing and measuring responses, and allowing the fab to dictate the shape of the operating curve. They may be sufficient for internal benchmarking exercises and productivity improvement measures, but ideally an accurate operating curve should be a snapshot of the actual behaviour of the fab so that engineers can get a true picture of the cycle time/utilisation relationship. Another possible alternative when modelling the fab and attempting to generate operating curves is to use discrete event simulation (DES) modelling.

2.5 Discrete Event Simulation Modelling

DES models are models that evolve over time (dynamic), have statistical random (stochastic) inputs and outputs, and are concerned only with discrete instantaneous events. For the purposes of this thesis, the terms ‘simulation’ and ‘simulation models’ will henceforth refer to discrete event simulations and models.

A discrete event simulation model is usually analysed numerically with the aid of a computer. Quite often the set of experiments may require a statistical framework and almost always it is necessary to analyse the output using statistical methods.

2.5.1 Steps in a simulation study

Fig. 2.7 gives a more comprehensive description, (but still very brief in context of the discipline), of the steps in a successful simulation study, based on the work of Banks et al.

(2004); Carson (2005); Law and Kelton (1997); Law (2008); Nordgren (1995); Robinson and Bhatia (1995); Robinson (2003); Sadowski and Grabau (2004). It is worth noting, despite the distinct nature of the steps, they are not independent of one another and not necessarily carried out in the order given. There may be some overlapping or concurrent steps, particularly in the planning and conceptualising phases.

Problem Formulation: Once a decision has been made to use simulation, the first step is to outline the goal of the study. The goal is the single most important objective that must be accomplished for the study to be deemed a success. The goal may be a hypothesis-like question or ‘what-if’ scenario pertaining to the real system, or it may be something less defined, like increasing the knowledge about the system under investigation. More often than not though, it is better to have a clearly defined goal for the project.

Set objectives and overall project plan: The objective of a simulation study is usually to investigate some ‘what if’ scenario such as, ‘what is the resulting average cycle time gain if an extra machine is added to a particular machine group?’. Objectives such as these tend to define the boundaries and the scope of the simulation model. For example, if one wanted to examine ‘the cycle time impact on a machine group’ then the most essential components that need to be modelled are the machines in that machine group and lots that pass through the machine group. Aside from these essential components, other peripheral components may need to be incorporated such as operators, downtimes or setups. The necessity of these peripheral components is very much dependent on the objectives of the model and their relevancy to the main components. If operators are generally always available at the machine group and don’t have a large impact on the cycle time of lots in that area, it may not be necessary to model them. This assumption, provided a case can be made for it, is perfectly valid in terms of the objective of the simulation.

Model conceptualisation: It is generally recognized that conceptual modelling is

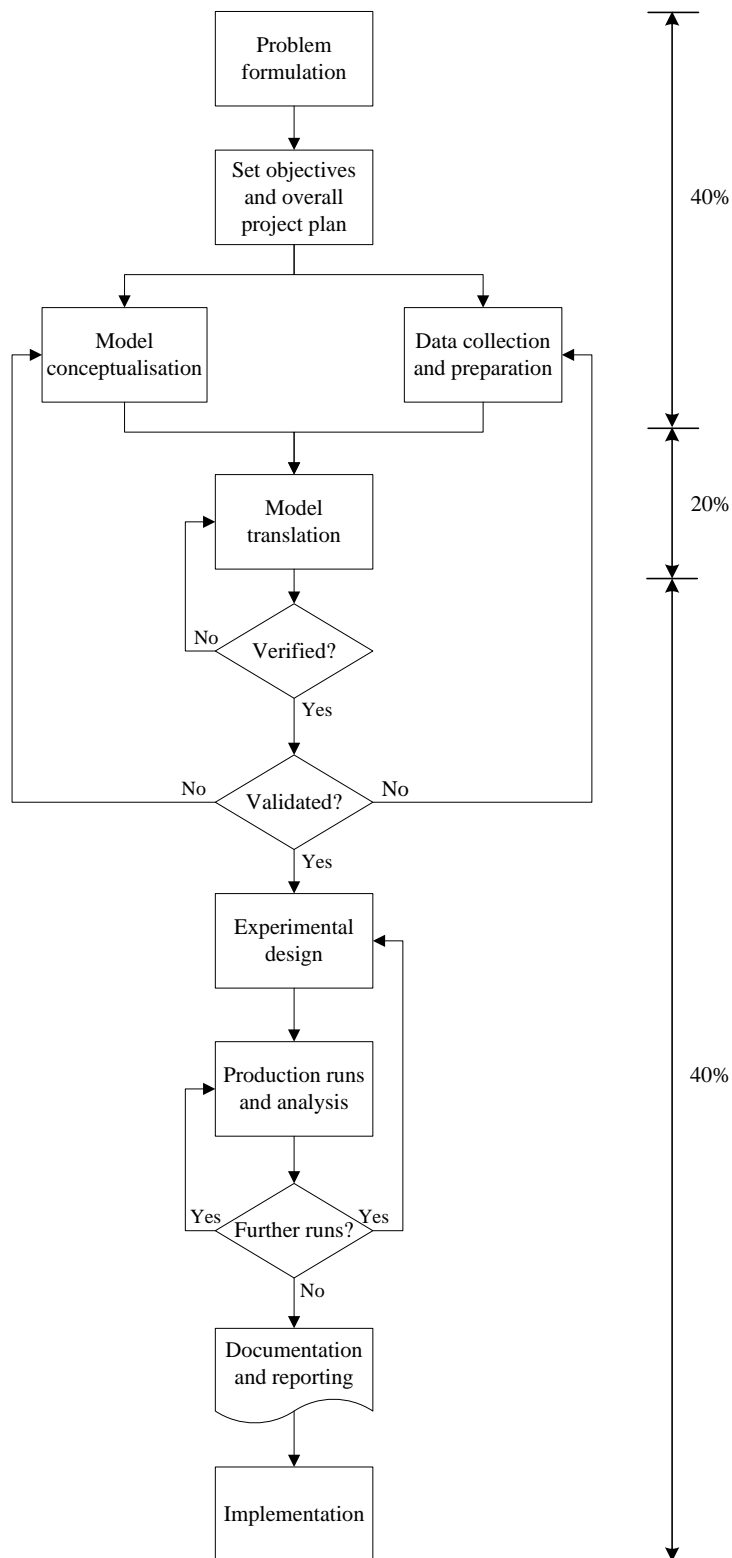


Figure 2.7: Steps in a simulation study (Banks et al., 2004; Carson, 2005; Law and Kelton, 1997; Law, 2008; Nordgren, 1995; Robinson and Bhatia, 1995; Sadowski and Grabau, 2004).

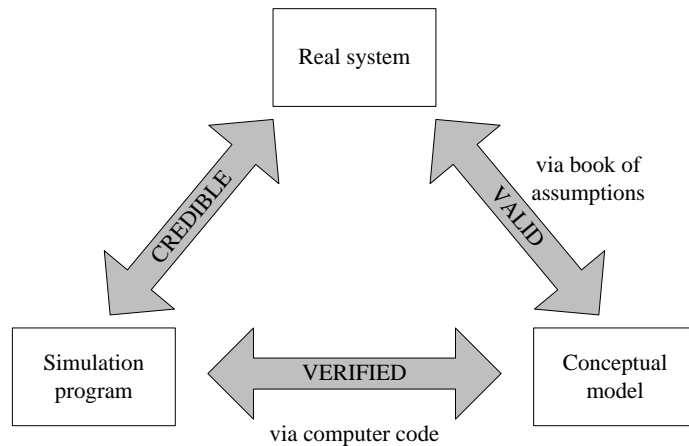


Figure 2.8: Phases of a simulation study.

one of the most vital parts of a simulation study (Robinson, 2006; Robinson and Brooks, 2010). DES modelling has the ability to capture and model a system to any realistic level of detail. This makes it a very powerful tool for engineers and system designers. However, the power and capability of simulation models is often made redundant by the lack of meaningful and correct implementation of proper modelling techniques. Unfortunately, a common misinterpretation is that simulation is an exercise in computer programming or computer model building. This is not the case; the most important aspect should be the construction of the conceptual model. The conceptual model is an abstraction of the real system. It is created by making a number of assumptions about the real system that are generally driven by the goals or objectives of the simulation study. In the conceptualisation phase, the model's scope, scale and depth are defined. These parameters are based on the project objectives, performance measures, data availability, credibility concerns, computer constraints, opinions of SMEs and time and money constraints (Law and Kelton, 1997). How the model will interpret the inputs and the logic of the conceptual model are all declared in this step. Since there is no model that can represent a substantial piece of reality 100% then it must be that there is some translation between these two systems or entities. The translation is done via the 'book of assumptions' (see Fig. 2.8). As the modeller analyses the real system they make

certain assumptions about how it operates in order to place some rules around the system. These rules form the model logic.

Data collection and preparation: The old adage of ‘garbage in, garbage out’ applies to the data collection steps. If the input to the simulation is inaccurate or insufficient, the results will also be inaccurate, regardless of how ‘good’ the model is. Data collection, availability and credibility is usually one of the biggest detractors from carrying out a successful simulation project (Law and McComas, 1991).

Model translation: In almost all cases the model will be coded in a computer program. Entry level to building simulation programs has eased over the years with the advent of graphical simulation packages that are user friendly and do not require any knowledge of programming or computer code.

Verification: “Verification is concerned with determining whether a conceptual simulation model has been correctly translated into a computer program” (Law and Kelton, 1997). In other words, the modeller ensures that the actual program logic and coding is representative of the conceptual model and operates in a manner that is similar to that intended by the model. Whitner and Balci (1989) gave a comprehensive list of verification techniques that are outlined in Table G.1 on pg. G-1.

Validation: Validating a model “refers to the processes and techniques that the model developer, model customer and decision makers jointly use to assure that the model represents the real system (or proposed real system) to a sufficient level of accuracy”(Carson, 2002). Law and Kelton (1997) define validation as the “process of determining whether a simulation model... is an accurate representation of the system, for the particular objectives of the study”. The added phrase “for the particular objectives of the study” implies that the model need only represent the parts of the real system of interest or the portions that have an impact on the parts of interest. A common technique known as a *structured walk-through*, whereby the

key parties get together and discuss the list of modelling assumptions, is often used to validate models. Other validation techniques are listed in Table G.2 on pg. G-4.

Experimental design: The experimental design is a key aspect of a successful simulation study. It may be necessary to make some pilot runs or preliminary calculations to determine the simulation run length, warm-up period, number of runs and replications, for each of the experimental scenarios. If the project involves comparison between alternative scenarios or configurations it may be necessary to implement a statistical comparison method such as a paired t-test or an all-pairwise comparison to help select the ‘best’ configuration.

Production runs and analysis: This step involves running the program and analysing the output data to ensure it is statistically sound. Typically, performance measures such as cycle time, throughput time or WIP levels are used as a trace metrics for the performance of the system.

Further runs or replications: Further statistical methods are used to investigate whether the output is sufficient. If not, subsequent replications and/or runs may be required.

Documentation and reporting: The majority of documentation and reporting, be it in-house, to clients, management, or to an academic audience via papers, journals and conferences, will most likely take place at the conclusion of the study. However, it is important to start any documentation from the beginning of the study. Gass (1984) recommends creating four main documents; the analyst’s, the end-user’s, the programmer’s and manager’s manuals. Banks et al. (2004) categorised documentation into two types; program and process. The *program* documentation keeps an account of how the model was coded so that subsequent users can understand how the model works. *Process* documentation involves a journal-like recording of chronological events that happened throughout the history of the simulation study

including the work done and all the decisions taken. This is similar to the ‘assumptions’ document discussed previously.

Implementation: At this point the project’s outcomes and goals should have been assessed and the hypotheses delivered from the results should be recommended for implementation. If on completion of the project, there are still some key parties in management or clients that do not favour the outcome or are reluctant to implement the recommended changes, then it may be necessary to return to the documentation and reporting step to drum up support and increase the credibility of the model (Law, 2008).

Fig. 2.7 also shows the relative ‘simulation effort’ that should be applied to the phases of the study. Law and Kelton (1997) recommend that the ‘40-20-40 rule’ should be used; where the first 40% applies to the planning, conceptualising and data gathering phases; the middle 20% applies to the actual model coding or programming; and the final 40% refers to the analysis of output. According to Law and Kelton, this is typically not the case, and often a highly disproportionate amount of effort is placed on coding or building the model. This is usually the result of a lack planning during the conceptualisation phases, or an insufficient understanding of the system by the model builders. In such a case, the modeller will often have to take too many breaks from coding to return to data gathering activities which can inhibit the success of the overall project.

2.5.2 Components of a DES model

The term ‘system’ has come to mean so many things that it is difficult to put a strict definition on the term. However, it should suffice to use the definition promoted by Gordon (1977), which states that a system is “an aggregation or assemblage of objects joined in some regular interaction of inter-dependence”. This thesis will refer to the ‘real system’, as some aggregation or assemblage in the real world that we are attempting to model or capture in a model. The term modelling unfortunately refers to both the

process of building the computer simulation and also constructing the conceptual model and as a result there can be some confusion. For the remainder of this section though, the term modelling will refer to creation of the conceptual model.

In a system there are distinct objects that form part of the system and these objects interact with each other to change the state of the system. These objects are usually referred to as *entities* and a piece of information attached to an entity is known as an *attribute*. An *activity* is a process that changes the state of the system and takes a period of time. An *event* is an occurrence in the system that happens instantaneously and also changes that state of the system (Banks et al., 2004; Carson, 2005; Gordon, 1977; Law and Kelton, 1997; Pidd, 1992; Schriber and Brunner, 2010). For example, if the system being analysed is a manufacturing system, then an entity could be a part or widget, an attribute of that part could be its due date, and an activity could be some welding operation that the part requires. An event could be the shipping of parts or the arrival of raw materials. There is some overlap between the terms *event* and *activity* and requires further clarification - an event is the result of an activity and can only happen if an activity occurs.

Once the components of interest in the real system have been categorised, it is necessary to encode them in a computer program. Graphical general purpose simulation packages are becoming increasingly popular and offer an accessible way to build simulations without the need for any programming expertise. These packages offer a range of *blocks* that can be placed into a canvass window, (examples of such software include ExtendSim, Simul8, PlantSim and Witness). The blocks can then be used to represent common real system elements.

Jeong et al. (2009) further discriminates between entities by using the term *resources* and *locations*. *Resources* refer to objects that provide a service, whereas, *entities* are the objects that circulate the system and receive the service by requesting a *resource*. If the *resource* is unavailable, then it joins a *queue* and waits for one to become available. A *location* component is typically used to refer to a *resource* that is 'static'. For example,

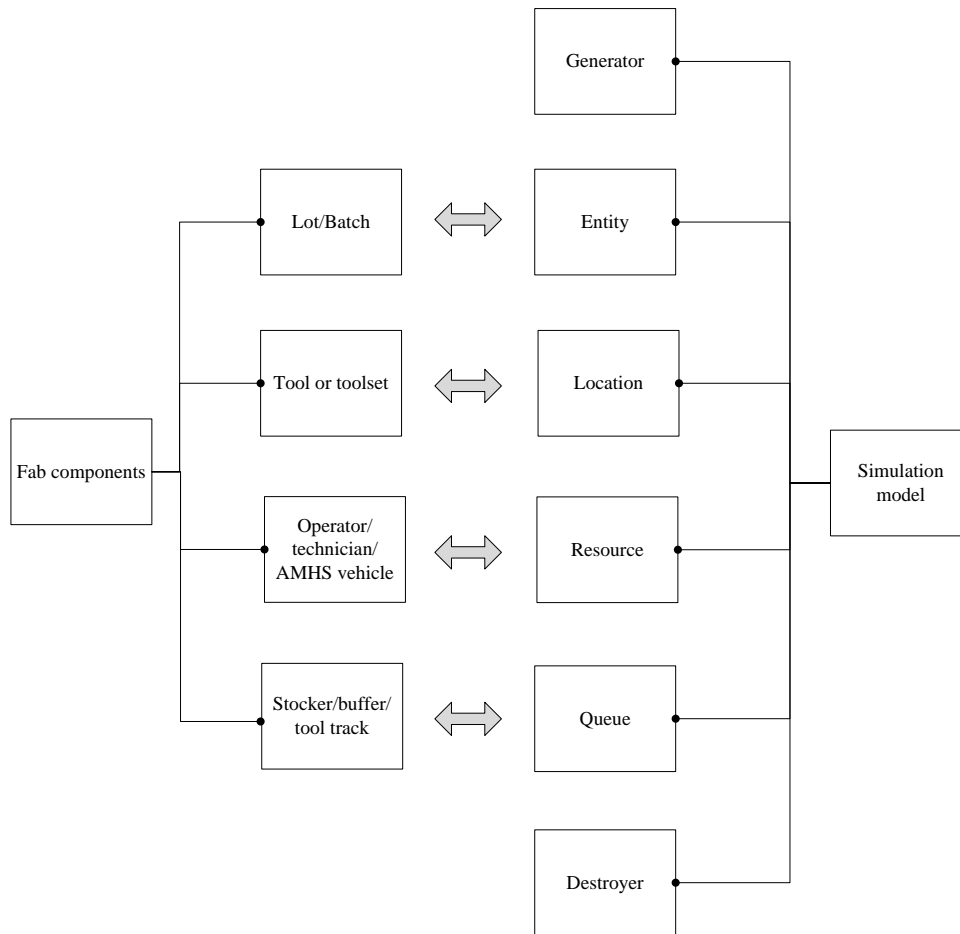


Figure 2.9: Components of a semiconductor fab categorised using simulation model components suggested by Jeong et al. (2009).

Fig. 2.9 shows how the components of a semiconductor manufacturing system might be categorised in a DES model. A machine or toolset might be represented as a *location*, due to its fixed positioning, whereas an AMHS vehicle or a maintenance technician might be referred to as a *resource* as they are not in a fixed location. Also, the wafer lots and batches are shown as *entities* as they circulate the system and require the service. *Entities* enter and exit the system using *generators* and *destroyers* respectively, which generally signal the bounds of the model outside of which, the modeller is not interested.

2.5.3 Justification for using DES modelling

Simulation modelling of semiconductor manufacturing plants is a topic of great interest and is much commented on in the associated literature. Banks and Norman (1995) stated

that “many companies are discovering that the value of simulation software goes beyond its ability to offer a peek into the future. It has numerous other benefits, including its ability to help managers make better decisions, explore possibilities, understand why certain phenomena occur, identify constraints and diagnose problems”. The feeling in the industry according to Atherton and Atherton (1995); Miller (1990); Rubinstein and Melamed (1998) is that simulation is the best approach for modelling semiconductor fabs. This is also true for generating operating curves, Fowler et al. (2001) stated that “as the system increases in complexity, simulation analysis becomes the most viable approach for generating the curve”.

Carson (2005) suggested several situations when simulation is most useful. These scenarios, are listed below and a case is made for each point, regarding the use of simulation to generate operating curves in this thesis.

1. *There is no simple analytical solution available or such a solution does not offer the required accuracy.* The reasons for not using analytical models when attempting to generate operating curves for semiconductor fabs is outlined in detail in Section 2.4.
2. *The real system under investigation can be captured, i.e., it is possible to build a logical interpretation (conceptual model) of the system that describes the real system to a required degree of accuracy.* This is possible, but a non-trivial task, as was shown by Boning et al. (1992) and Sprenger and Rose (2010), who enabled their models by capturing the dominant structures and phenomena of a semiconductor manufacturing system and conceptualised them into modelling components. The development chapters in this thesis discuss the building of a conceptual modelling framework for semiconductor wafer fabs.
3. *If the system is new or not yet built, or requires major configuration changes that will have a significant impact on the system.* The assumption here is that the overall system remains relatively fixed but can change within the boundaries of the specification, e.g., the addition or removal of a tool from a toolset or changing the number of maintenance technicians.
4. *The changes to the real system being considered require significant investment and demand a high probability of success.* The purpose of this thesis is proof of concept rather than an actual investment analysis.

5. *Some forum is in place (or can be created) where the simulation team and all other parties including management, clients and people in the real system being modelled, can communicate easily and discuss and agree on the assumptions documents.* The stakeholders in the models built to generate operating curves partook in regular meetings to discuss the project, though no official forum was put in place. It is worth noting however, that such a collaborative environment may be conducive to system learnings outside of the scope of the modelling project. This was shown by Potti and Whitaker (2003), who used their model as the focal point for all communication between fab departments regarding productivity improvement projects.
6. *There is some type of animation available. Animation increases the chances of a more credible simulation model that is understood and trusted by those who have invested in it and also the end-users.* Implementations of the DES models in this thesis were created using ExtendSim, a graphical simulation modelling tool capable of both 2-D and 3-D animation as well as basic ‘proof animation’. Some of the models were built in SimPy, a library for Python. These models do not support animation but Python has a number of libraries including *Pyglet* and *Pygame* that could be used to animate the SimPy simulation models.

Many of these recommendations were also discussed by Banks and Gibson (1996, 1997a) and listed below. Again, a justification for using simulation modelling in the context of this thesis is offered in Table 2.1.

Banks and Gibson further stressed that simulations do not provide an optimal solution, that is, they cannot recommend a system configuration that the analyst does not specifically investigate. This is a valid statement but does not interfere with the aims of the methodology in this thesis, which uses graphical comparisons between resultant operating curves to analyse various configurations. The assumption is that the user of the modelling applications will be knowledgeable about the system, and can offer alternate configurations, test them, and assess them based on their impact on the operating curve output from the model.

When using simulation, the advantages tend to lie in the areas of general applicability and capability. Typically, most complexities can be modelled, the only limiting factor is the cost and time-frame of the project. Another benefit to using simulation is that it is possible to model transient behaviour. Klein and Kalir (2006) discussed this type

Table 2.1: Justification for using simulation modelling to generate operating curves in semiconductor manufacturing, based on the recommendations offered by Banks and Gibson (1996, 1997a).

Circumstances	Justification
A common sense analysis is available.	Capturing semiconductor manufacturing systems in detail is a non-trivial task and it is highly unlikely that a common sense analysis is available that can capture the complexities sufficiently to generate an operating curve.
An analytical solution is more appropriate.	Section 2.4 details the reasons why an analytical modelling approach to generating operating curve for semiconductor fabs is alone insufficient.
Direct experimentation with the real system is easier.	This could be very costly in a fab, and have a negative impact on production targets.
The simulation costs exceed the rewards.	The models generated in this thesis are proof of concept. However, the modelling strategy implemented was designed to minimise the labour involved in generating an operating curve via simulation models. Hence a relatively low cost would be required to generate a curve which may be highly valuable to the fab.
Simulation resources and expertise is not available.	The models and applications developed in this thesis are tailored to use by non-simulationists.
There is insufficient time to perform the simulation analysis.	The models and applications arising from this thesis are fully automated requiring very little analysis time.
There is no data available.	It is assumed that the programs and applications have access to factory data which is electronically stored. This is generally a reasonable assumption in highly automated semiconductor manufacturing systems where an abundance of data is recorded.
The model can't be verified or validated.	A full validation and verification of all models is performed.
The systems behaviour is too complex to be captured.	The highly complex nature of semiconductor manufacturing will be modelled by modularising its most common aspects into repeatable and reusable DES models.

of modelling and the benefits of using it to monitor a fab undergoing *ramping-up* to a new product. For these reasons simulation is currently leading the line for modelling semiconductor manufacturing.

In fact, some simulation models may become such an important tool for factory management that it can drive most, if not all of the decisions made. An example of this was shown by Potti and Whitaker (2003), who used their simulation model as the focal point for all communication between fab departments regarding productivity improvement projects.

2.5.4 Flexible reusable DES modelling

As outlined previously, simulation modelling is regarded as an appropriate tool for modelling complex systems such as a semiconductor manufacturing system. However, the disadvantages of using simulation lie in the amount of time, effort and resources required to bring a simulation project to fulfilment. Some of the research into simulation modelling of semiconductor manufacturing (e.g., Ehm et al. (2009); El-Kilany (2003); Mackulak et al. (1998); Paul and Taylor (2002); Pidd (2002)) has focused on flexible and reusable simulation development and deployment, as well as standardisation of simulation frameworks and modelling inputs. Such efforts aim to create a full framework for a simulation project and allow end users to jump straight to the experimentation stage of a project without undergoing the time-consuming stages of data extraction, model building and coding. This type of modelling strategy appears to be ideal for generating fast and accurate operating curves.

Furthermore, the selection of simulation modelling over analytical modelling does not mean that analytical models do not play a part in the framework proposed in this thesis. Tractable analytical models will be used to recommend simulation run lengths, optimum design points and to predict the optimum number simulation runs or replications based on the work of Hoad et al. (2008, 2009). Combining these two facets; flexible-reusable modelling and automation of the design of experiments using statistical techniques and

analytical models, means that generating accurate operating curves can be fast, reliable and efficient.

When creating a flexible reusable model it can be more difficult to assess the boundaries, scope and scale of the model. A important opportunity can often be missed that could potentially make the simulation model far more useful. To return to previous example given in Section 2.5.1. What if sometime after the initial project, management return to the simulationist with the task of optimising the number of operators that tend to the machine in question? At the time of the initial simulation project, the subject of operators was deemed irrelevant and outside the scope of the investigation. The question then becomes whether it is easy to incorporate operators into the model structure without requiring a complete rebuild.

Another possible scenario; it is noticed that the machine group under investigation is very similar to another machine group at different location within the factory. Is it possible to adapt the model, or is it restricted by the original coding or build? Also, what if management want to see the effects of adding or removing multiple machines from either machine group, is this also easily managed? All these questions define the reusability of the model. Reusability and flexibility are interchangeable terms, in that, a model is designed to be flexible for the purposes of making it reusable, and the targeted level of reusability dictates the level of flexibility of the model. These potential issues should be addressed when scoping the initial project and some foresight may be required on the part of the project planner.

For example, in order to encompass many of the complexities of semiconductor manufacturing, most simulation models in the industry are large and complex. This level of complexity is usually a consequence of the demands of management. Often there is a desire to model too many aspects of the fab, and as a result, modellers draw the bounds of the simulation scope too wide (Chwif et al., 2000; Law and Kelton, 1997; Sadowski and Grabau, 2004). This is typically one of the biggest pitfalls of a simulation project, because the modeller spends too much time trying to capture every detail about the

system and the project becomes an endless exercise in computer programming.

An inexperienced simulation practitioner building a reusable model may try to incorporate too much or expand the scope of the model too wide. This problem of poor scoping of the model is generally as a result of the fear of ‘leaving something out’ that may be needed at some point in the future. The result being a slow, inefficient and cumbersome model with excessive detail, which also requires a very high level of cost and effort to create and maintain. This can have serious implications to the overall project and may put it beyond its budgeted timeframe and cost.

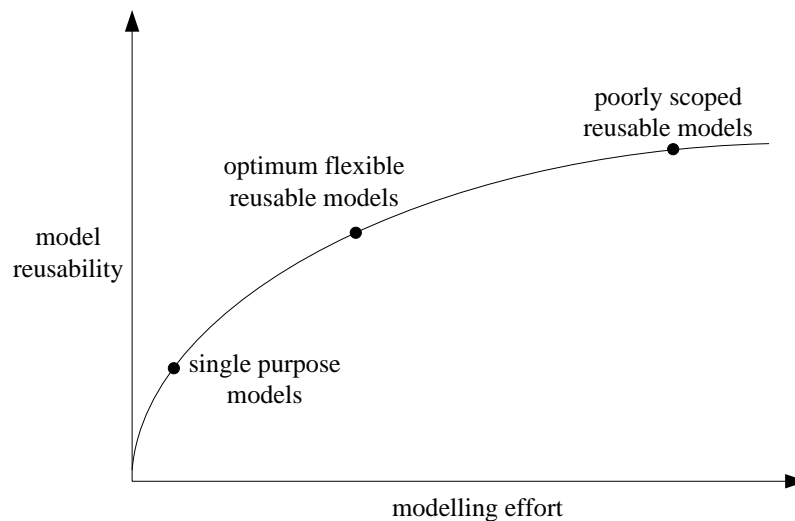


Figure 2.10: Modelling effort and model reusability.

An optimally scoped reusable model, such as that shown in Fig. 2.10, need not capture every component or constituent of the real system. It should have sufficient level of detail to capture the relevant phenomena that concern the current objectives and sufficient breath of design and flexibility that it can successfully incorporate future components without a complete rebuild. This means that a reusable model should have sufficient scalability and flexibility to achieve the stated objectives of the project. Both of these goals can be achieved if a carefully planned and mindful approach to reusable modelling is used during the initial stages of planning.

Hence it may be necessary to expand upon the objectives of the simulation study. The more immediate hypothesis-like objectives such as ‘what is the resulting average

cycle time gain if an extra machine is added to a particular machine group?’ may need to be expanded to include more open objectives such as ‘what is the cycle time impact to lots subject to competition for resources?’, (where the resources might include operators, machines or transportation systems). More open statements or objectives encourage more open-minded decision-making during the initial planning phases. An example of this can be seen in the work of (Johansson and Grunberg, 2001). They noted that a flexible reusable modelling strategy should be the focus throughout the fulfilment of the modelling project, meaning that making the model reusable was not restricted solely to the model build phase, it should be incorporated across each phase of the simulation study.

This reusable modelling strategy can also aid modellers in selecting the software tools or packages that they use. A common pitfall is selection of software with insufficient functionality. Banks (1999) calls this the 90% syndrome, whereby you find that the software has sufficient rudimentary capability to achieve 90% of the original objective, and an extra 2-5% can be achieved or eked out by using the software’s functions in unusual and unorthodox ways. Finally, it becomes apparent that a 100% complete solution is outside of the capabilities of the software and a complete change of simulation tools is necessary. This can be avoided if the original objectives include some foresight about possible future functionality of a reusable model, and if proper research is conducted into the capabilities of the modelling packages available.

The commercial software packages used by simulation modellers has seen a similar shift in the flexible and reusable trend, with the more popular software providing modularised ‘off the shelf’ sub-models (modelling blocks) that can be positioned to model real system entities (Valentin and Verbraeck, 2002; Verbraeck and Valentin, 2008). This reduces the time spent coding the program and allows the user to focus more on the decision-making aspect and getting the conceptual model correct.

Such techniques can have significant benefits when deployed to semiconductor manufacturing. The basic entities that constitute a fab are similar and/or repetitive and hence,

the same modelling blocks can be used or reused, as was shown by Boning et al. (1992); El-Kilany (2003); Sprenger and Rose (2010). For example, custom sub-models or modelling blocks can be used to represent the workstations in a fab, thereby creating a set of nodes within the model through which the material flows. Creating a full factory model, (if sufficiently capable sub-models are available), should then be a matter of structuring sub-models in an appropriate fashion. Such rapid modelling methods appear to be the most efficient route of generating simulation based operating curves for a semiconductor fab.

Automated generation of simulation models

If the real system consists of common components that can be identified and compartmentalised, then it is possible to completely remove a simulationist from the process of building models and create programs to generate simulation models. For this, it is necessary to build generic models consisting of components that are robust enough to make the model applicable to a large range of inputs and systems. Steele et al. (2002) outlines some of the basic requirements of a generic reusable model,

- *Ensure that the important factors or components of the system are included.* This helps to define the scope of the models and reduce the system complexity,
- *Simplify the input.* *The input data should be easily interpreted and well defined.* This emphasises the use of system descriptors such as the Sematech *Semiconductor Wafer Manufacturing Data Format Specification* (Feigin et al., 1994) or information models such as the Core Manufacturing Simulation Data (CMSD) (Riddick and Lee, 2008),
- *User-friendly output with graphs and charts.* This further promotes the use of operating curves to analyse the results from the model.

Building a generic simulation model requires more time and effort in the initial phases, however, it can have a significant pay-off in the long run. Linking the model to the input data generally requires some program or framework to assemble the data, populate the generic flexible model and control its execution and output. Mueller et al. (2007)

presented a framework for generation of models based on the *Semiconductor Wafer Manufacturing Data Format Specification* using an object-orientated Petri-net interpretation of the specification. Mueller et al. listed the advantages of the method as:

- The end-user did not have to do any coding,
- The simulation generation is a rapid process,
- The model generation is fixed, thereby removing any chance of coding or programming errors,
- Theoretically, there is no limitation to the size and the scale of the model.

Automatically generated models, however, are not without some disadvantages. Bergmann and Strassburger (2010) discussed the challenges of such a modelling strategy and outlined them as follows:

1. Incomplete data in external systems: The core input data that drives the model may not always be reliable or it may not be possible to automatically capture the required information from the real system, that is, where operations are not monitored. Additionally, summary statistics and distributional information for activities may not exist.
2. Generation of dynamic/complex behaviour: The complexity of some systems may inhibit algorithmic translation and some dynamic behaviour may be lost.
3. Support of cyclic approaches involving multiple model generation cycles: Bergmann and Strassburger estimates that an automated model can only ever capture about 80% of that required, the rest needs to be manually added. These manual additions also need to be monitored and documented.
4. Support of multiple life cycle phases of the production system: Most real world systems evolve over time, capturing this evolution of the real system in an automated model can be a challenge.

These comments are valid and noteworthy, however, most of these challenges are not restricted to that of automatically generated models, and they are also issues that occur when performing one-off simulation models also. For example, the first challenge, data issues, exist regardless of the long-term aims of the simulation project. In other words,

if the data does not exist or requires treatment, then this issue affects both modelling strategies. One might argue that manually inputting the data for one-off models might be easier, however, if the data is in bulk, then it is usually necessary to create a script or program that can prepare the data, which promotes the case for reuse of this script in a reusable generic modelling project.

On the second point, system complexity will affect both one-off and reusable auto-generated models. Understanding a system and translating it to algorithmic form (conceptual modelling) may be a more difficult task in a generic model, that must encompass more 'behaviour', however, the learnings gained could have a pay-off in terms of greater understanding of the system. For example, the one-off model may incorrectly disregard system behaviour that is deemed less relevant, whereas, the larger scope of a flexible reusable model is less likely to disregard import behaviour.

The third challenge, according to Bergmann and Strassburger, refers to the impossibility of completely automating a simulation model. While this is a fair observation, it should not mean that it should not be attempted. Finally, the last point expresses concerns over capturing the evolution of a system in an automatically generated model. However, here it is likely that having an automated model that captures a previous iteration of the real system is a good starting point for modelling its current state. With these challenges in mind, the benefits of having an automatically generated simulation model far outweigh its disadvantages.

2.6 Summary

The following summarises the findings in the literature review;

- Operating curves are a very important, if not the most important, metric for a semiconductor fab.
- Analytical methods for generating the operating curve for a semiconductor fab are flawed by the fundamental assumptions which they are based on; the independence of arrival & service pattern and the assumption of identical toolsets.

- Analytical or queueing models are also difficult to implement because they assume a fixed number of tools in a system and require single values for system variability, which can be hard to retrieve from a real fab. As a consequence, the queueing models themselves have been used to predict system variability, which is then fixed for all subsequent calculations. Due to this broad assumption, operating curves based on analytical models can only be used for benchmarking purposes, and there is no guarantee that they are a good approximation of the actual underlying operating curve.
- DES models offer an alternative method to generate the operating curve, and are not restricted by the same assumptions. However, the lead time to building a simulation model is long for such a complex system. This can be reduced by implementing a flexible modelling framework to auto-generate models of the real system. The design of simulation experiments can also be aided by using basic queueing approximations of the system to estimate the experimental parameters for the simulation models, as discussed in the following chapter.

An Automated Framework for Designing Discrete Event Simulation Experiments

All too often, much consideration is given to the creation of a simulation model with an emphasis on the idea that if the computer model is ‘right’ then the simulation study will be successful (Law and McComas, 1991). This over-emphasis on the model building phase usually results in the sacrificing of a proper framework for the simulation experiments and a lack of application of the correct statistical and scientific procedures. This chapter describes a framework for automated design of simulation experiments that hopes to streamline this process and enforce due diligence in simulation projects. The framework relies heavily on estimations and approximations from queueing theory.

It is worth emphasising that the queueing models and approximations are being used

merely as guides for the simulation models as opposed to accurate estimators for the system under study. This means that special dispensation is given to the use of queueing models, when they are not guaranteed to be applicable for the system under study. However, considering that some common sense scientific methodology is required to construct an automated framework, it is best to use potentially inaccurate general queueing approximations rather than nothing.

Throughout the chapter there is a description of the techniques and algorithms used to automate the framework. All of the accompanying source code can be found in Appendix A.

3.1 Simulation Effort

The concept of a minimum required effort for a simulation study is an inescapable aspect of simulation. A simulation is a series of logical controls that are tested by inputting a range of random inputs, and interpreting the random outputs. Therefore, it is necessary to use a large enough range of inputs so that the output range is similarly large, and some sort of statistical or stochastic pattern can be found. This stochastic pattern then allows ‘proper’ inferences or conclusions to be made about the logical controls (which collectively constitute the model).

3.1.1 Selection and location of design points on an operating curve

In order to construct an accurate operating curve with the minimum amount of simulation effort, a sufficient minimum number of design points on the curve are required. One solution is to select a large amount of design points and run the necessary simulations. However, in reality simulation effort is not ‘free’ and there is some cost involved. Whether that be the analyst’s time or more likely, the computer processor time required to run the simulation program, it is generally not feasible to plot a very large number of design

points. Hence, some trade-off between accuracy and simulation effort is required. Johnson, Leach, Fowler and Mackulak (2004) described this problem by assuming that there is a fixed budget of simulation effort available and the choice of location and quantity of design points could be represented as some function of the variance of the operating curve.

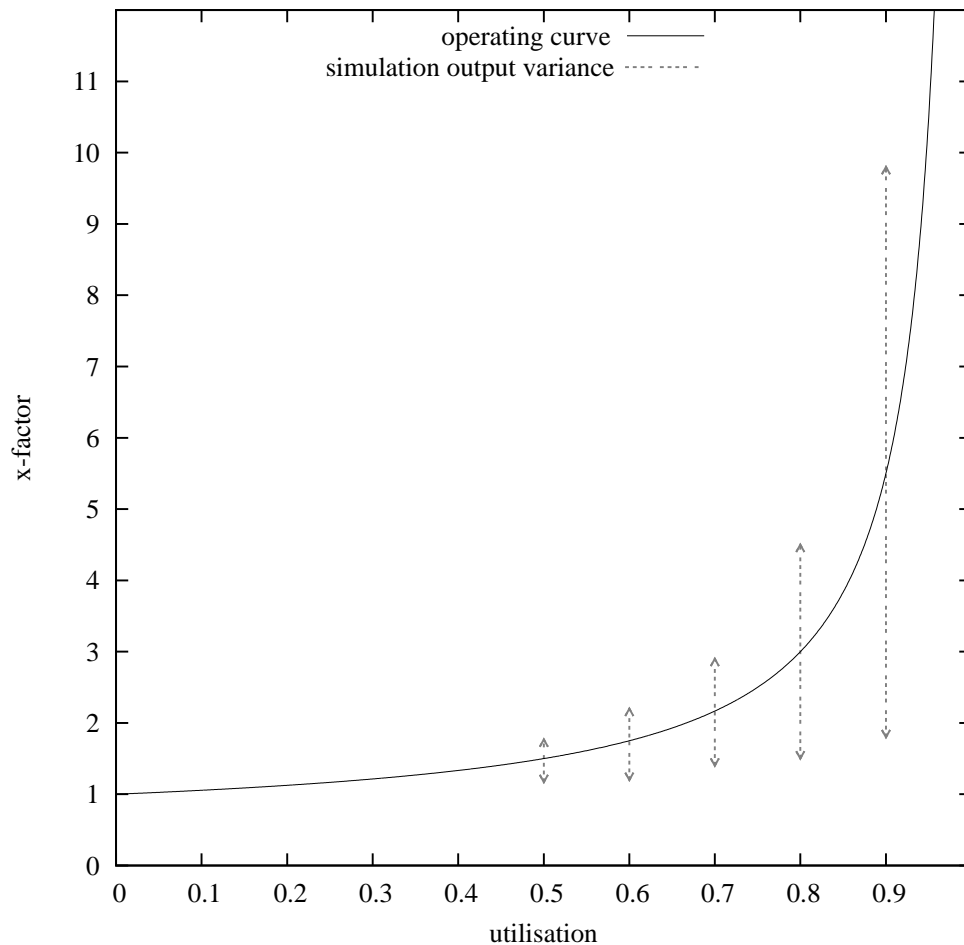


Figure 3.1: Operating curve indicating the simulation output variance.

They stated that the level of simulation effort exhausted could be directly proportional to the curve variance, meaning that the greater the variance from a more heavily utilised system, the more simulation effort that is required. To calculate the required simulation effort Johnson, Leach, Fowler and Mackulak (2004) used weighted ratios based on the variance at each point such that the corresponding absolute confidence widths were equal.

While this technique, (also discussed by Fowler et al. (2001, 2008)), provides a log-

ically sound methodology for the allocation of simulation effort to design points, it is somewhat insufficient on two accounts. Primarily, there is no guide for location of the design points, the assumption is that the analyst selects the design points. This appears to be a very arbitrary element of an otherwise scientific technique. Secondly, as can be seen from Fig. 3.1 the variance increases at a very fast rate as utilisation increases. This means that design points on the high side of the curve require a very large amount of simulation effort. The level of effort required to capture a system with a higher loading than it rarely operates at, may be very time consuming with very little reward. Johnson, Feng, Ankenman and Nelson (2004) echoed this statement by saying “the highest throughput level tends to consume nearly all of the simulation effort”. They also stated that “. . . often, the lower design points receive virtually no simulation effort, requiring the analyst carrying out the simulation to give the design points a minimum default value. . . . (For example,) at the following throughput levels: 60, 70, 80, and 90 percent, the 90 percent design point claims approximately 99% of the budget available for the simulation effort”.

Clearly there is too much emphasis placed on the high and volatile portion of the operating curve. Much effort is required to capture the operating curve in this area sufficiently, but real systems rarely venture into this region, at least not for long periods of time, and it may be more beneficial to allocate effort to the more likely areas of operation. With this in mind, a technique is proposed of locating design points based on their proximity to the location of an optimum operating point on the system curve, where the greatest change in the curve takes place.

3.1.2 Allocating simulation effort

As can be seen from Fig. 3.2, there appears to be a very asymptotic nature to the shape of operating curves that becomes more apparent in systems that are configured better to handle increased traffic flows. For example, systems with a high number of parallel servers, or with low to moderate variability, can cope better with increased demand.

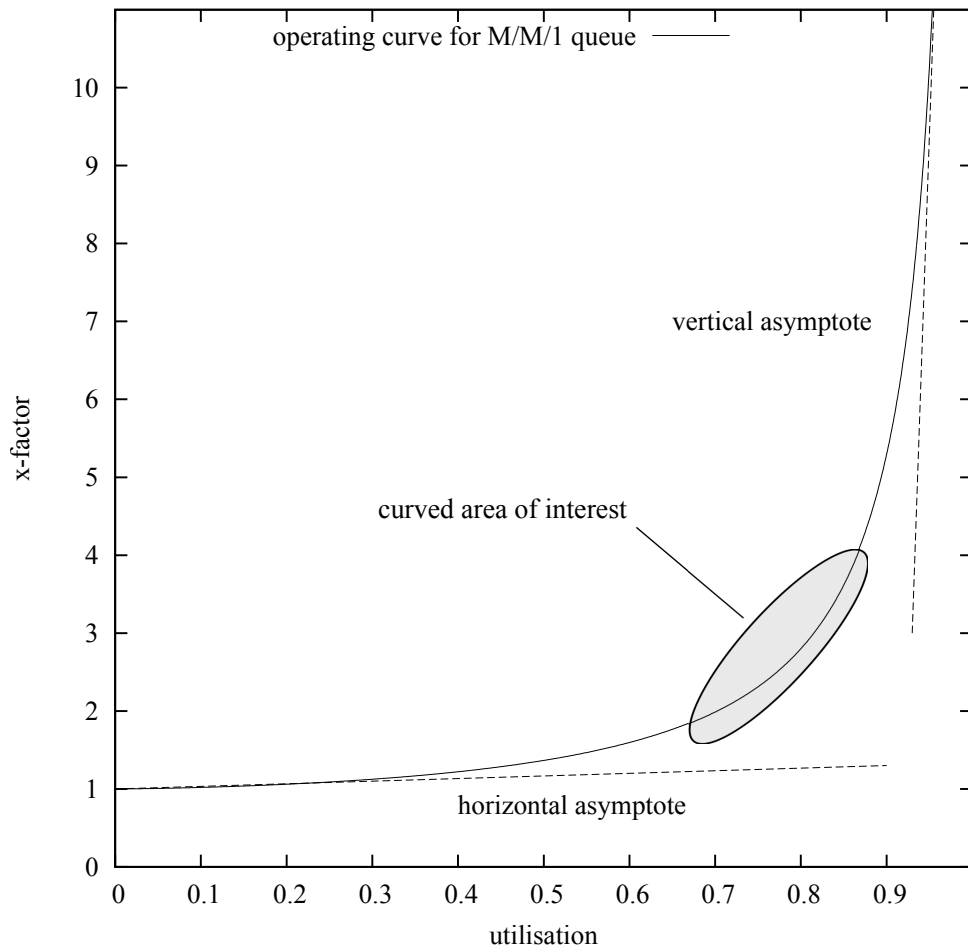


Figure 3.2: M/M/1 operating curve showing the ‘vertical’ and ‘horizontal’ asymptotes and the curved area of interest.

Therefore, they have a curve that exhibits two asymptotes; a horizontal one that encompasses the normal operational bounds when traffic intensity is so low it has little or no effect on the cycle time, and a vertical one that shows the critical level of traffic that the system can handle. Any level of traffic on the vertical asymptote will be generally unmanageable and would result in very high queueing and cycle times.

These two asymptotes become less pronounced as the system becomes less efficient, however, they provide the basis for estimating design points based on the areas of interest in the operating curve. Assuming that the asymptotes are approximately horizontal and vertical, it is not wholly necessary to allocate many design points to these ‘straight line’ areas, and it may be more beneficial to allocate the bulk of design points around the curvature. In order to investigate the curved areas, some definition of the level or rate of

curvature was required.

Allocating simulation effort using the level of curvature function

From calculus, the level of curvature $k(x)$ of a function $y = f(x)$ is given by Eq.(3.1) (Stroud, 1995).

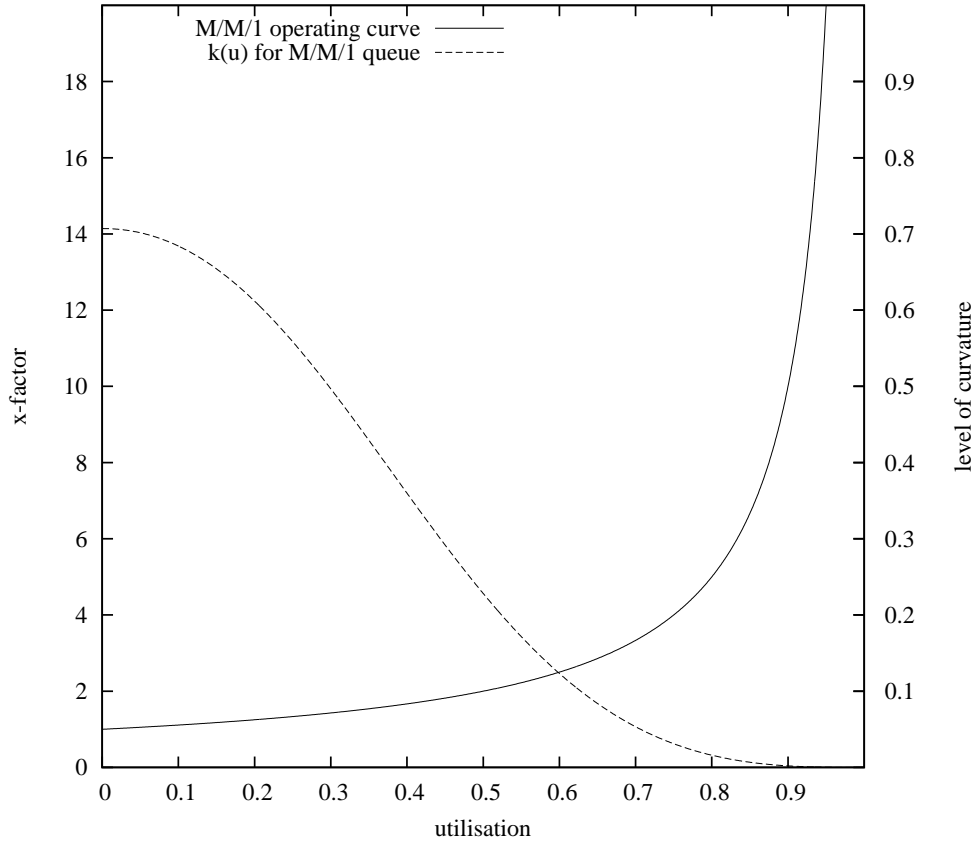


Figure 3.3: The curvature level $k(u)$ for an M/M/1 queueing approximation according to Eq.(3.4).

$$k(x) = \frac{|f''(x)|}{[1 + [f'(x)]^2]^{3/2}} \quad (3.1)$$

$$\frac{\partial (CT_{M/M/1})}{\partial u} = \frac{1}{(1 - u)^2} \quad (3.2)$$

$$\frac{\partial^2 (CT_{M/M/1})}{\partial u^2} = \frac{2}{(1 - u)^3} \quad (3.3)$$

Calculating the derivative (Eq.(3.2)) and the double derivative (Eq.(3.3)) with respect to u of the queueing approximation for the M/M/1 queue (with $t_e = 1$) from Eq.(2.3) on pg. 16, and substituting these values into Eq.(3.1), gives the rate of curvature as,

$$k(u)_{M/M/1} = \frac{\frac{2}{(1-u)^3}}{\left[1 + \frac{1}{(1-u)^4}\right]^{3/2}} \quad (3.4)$$

This method was found to be not directly applicable in these circumstances due to the very large differences in scale between cycle time and utilisation. Given that utilisation is on the range $(0, 1)$ and cycle time is on the range $(0, \infty)$, it requires some normalising of cycle time, before it can be used in these circumstances. A cursory examination of different normalising factors showed that the position of the highest level of curvature shifts depending on the normalising factor used. Any further examination of this phenomena was deemed to be beyond the scope of this thesis.

Allocation of simulation effort using the u/CT curve

An alternative method of approximating the area of most interest is to characterise how utilisation changes as cycle time changes, that is, map the u/CT curve. The u/CT curve, is derived by plotting u/CT against u , and is similar to what Ignizio (2009) calls the load-adjusted cycle time efficiency (LACTE) curve.

The u/CT curve is effectively a measure of the quotient impact of utilisation and cycle time across a full loading profile. Therefore, it allows one to analyse the utilisation level at which any increase in loading will result in a significantly disproportionate increase in cycle time or x-factor. In other words, it is effectively a ratio of the measure of the gains achievable by increasing utilisation before incurring a heavy cycle time penalty.

Figure 3.4 shows a u/CT curve for an M/M/1 queue along with its operating curve and the ‘envelope’ of operation for the u/CT curve. The envelope forms a triangular region, which is bounded by the linear function $CT = t_0$ (raw process time (RPT)) in the range $u = (0, 1)$ and $u/CT = 0$ at $u = 1$, meaning that the best possible curve is one

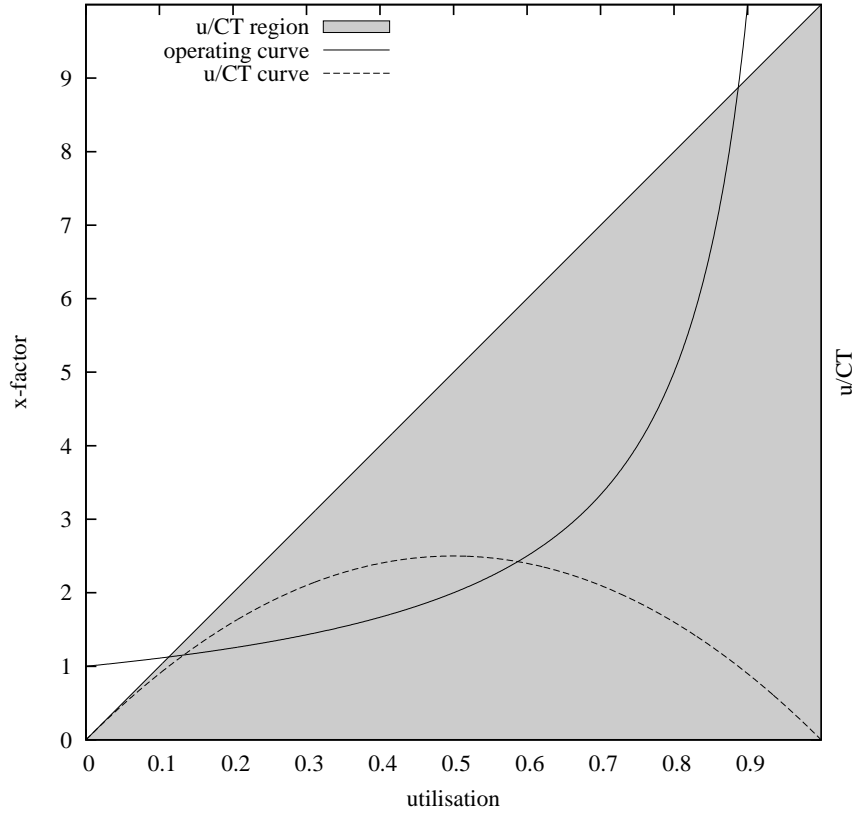


Figure 3.4: Operating curve and equivalent u/CT curve for M/M/1 queue, showing the u/CT region and an optimum operating point at $u = 0.5$.

that fits this region, that is, a system where u is strictly less than 1 and the cycle time is equal to the raw process time.

If the operating curve is monotonically increasing for u , then the u/CT curve will observe a single peak. This peak can be found by solving Eq.(3.5) for unique root u .

$$\frac{\partial (u/CT)}{\partial u} = 0 \quad (3.5)$$

$$\frac{\partial (u/CT_{M/M/1})}{\partial u} = 1 - 2u = 0, \quad (3.6)$$

From Eq.(2.3) which approximates the cycle time for an M/M/1 queue, assuming that the process time t_e is 1.0, (meaning that the x-factor is equal to the cycle time), then the peak of the u/CT curve is given by Eq.(3.6) where $u = 0.5$.

The location of the peak of these u/CT curves is of most interest to the user, and can

be used to identify the inflection region of the standard operating curve. By examining the operating curve, it can be seen that where the u/CT value is low, utilisation is either low or high. On the low side, the operating curve is generally quite flat, meaning that there is not much of a proportionate increase to cycle time as utilisation is increased. Therefore, it is not necessary to select many design points in this region, as the system is under-utilised and cycle time results are generally close to the RPT. It can also be shown that the variability of independent experimental replications of a performance measure in this region is also quite low (Johnson, Feng, Ankenman and Nelson, 2004). The other region where the u/CT value is also low is at very high levels of utilisation, where the impact to cycle time is very significant. The variance from this region is high and can require many replications.

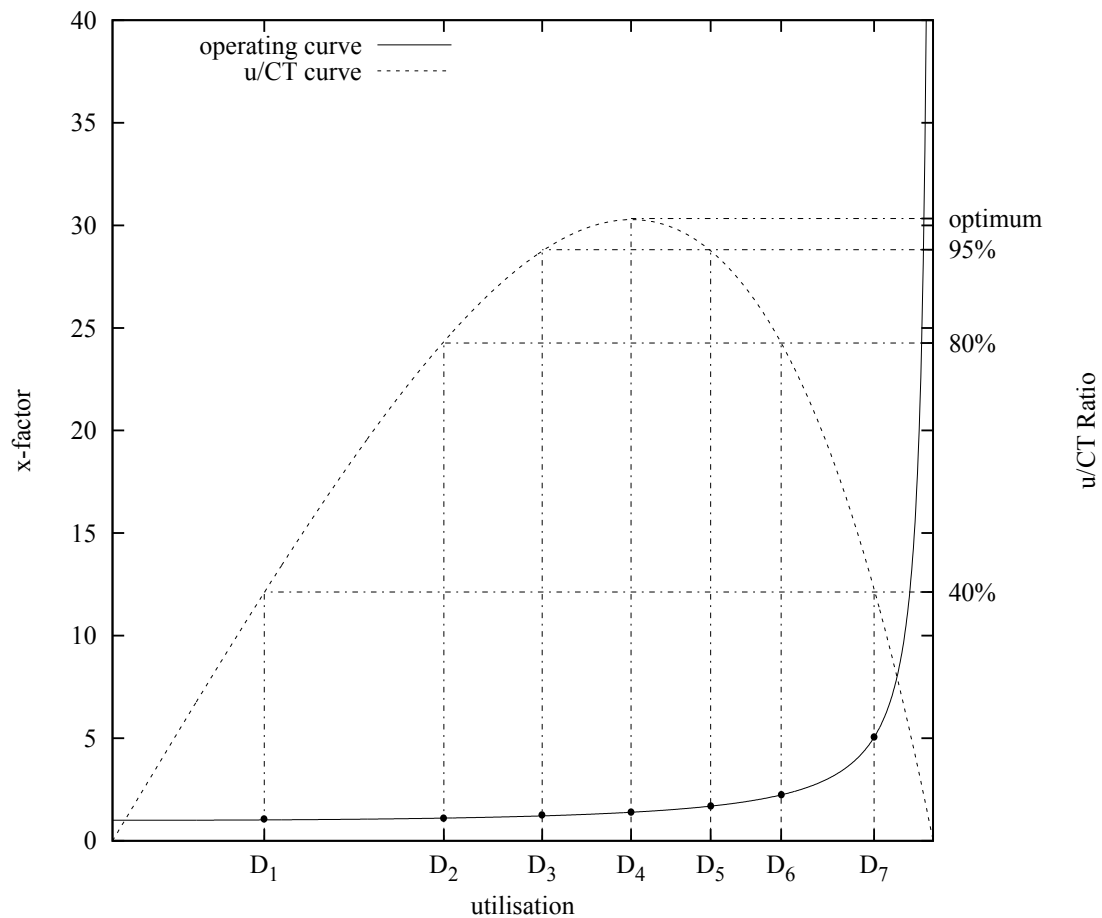


Figure 3.5: Design points for an M/M/3 queuing system at 100%, 95%, 80% and 40% of u/CT .

Hence, using the u/CT curve as a guide, will allocate a higher number of design points to the area of greatest curve change, while the areas of low curve change, such as the very high and very low utilisation areas, are not allocated as many design points.

Figure 3.5 illustrates this method for an M/M/3 queuing system. Assuming a fixed number of design points of 7, where one design point is the peak of the u/CT curve (where the operating curve has its inflection point) and the other 6 points form pairs on the u/CT curve. So, assuming that the values are selected at (0.4, 0.80, 0.95, 1.0), that is, 3 pairs and 1 at the peak, then the equivalent utilisation values are given by the set $\{0.185, 0.404, 0.524, 0.632, 0.729, 0.815, 0.923\}$.

This method is capable of determining more appropriate design points, however, there still requires some guess work and intuition about the values to choose on the u/CT curve. An alternative method is to use some form of probability selection. Given a fixed number of required design points, one could sample on the range (0,1) with a higher probability given to larger numbers. An algorithmic version of this selection policy could involve sampling values on u with a small interval of say 0.01, and its probability weighting factor could be its actual u/CT value. The main issue here is that with a very low number of design points being sampled, the probability of attaining many ‘unwanted’ values, or values that are grouped together is somewhat high.

Furthermore, this still does not solve the problem of selecting points that are too close together. Such weighted sampling methods are untested and for these reasons this thesis will rely on approximating values on the u/CT curve to construct the design points using the same method used in the G/G/3 queueing example from Fig. 3.5.

3.2 Method of Independent Replications

The output of a simulation is usually some performance measure of interest. Some of the performance measures most commonly collected for a simulation of a manufacturing system are cycle time, throughput or the number of items in a queue. These serve as good

performance metrics for the system because not only is the data of interest to engineers, but the trends formed by these values indicate whether a simulation run has ‘matured’ enough such that inferences can be made about the real system from the model.

The technique of performing a number of independent simulation replications requires assurances that a sufficient number of these runs has been conducted. A test for this, described by Law and Kelton (1997), requires a minimum number of replications n until,

$$\frac{t_{n-1,1-\alpha/2}\sqrt{\sigma^2/n}}{\mu} \leq \frac{\gamma}{1-\gamma}, \quad (3.7)$$

is satisfied, where $t_{n-1,1-\alpha/2}$ is the student t-distribution with $n-1$ degrees of freedom for a precision α , μ and σ are the mean and standard deviation of the replications respectively, and γ is the acceptable relative error. The accompanying code for this algorithm is given in Appendix A.1.

3.3 Whitt Simulation Run Length

From Whitt (1989b), for a G/G/m model, the required simulation run length is a function of the utilisation u , the squared coefficients of variability for process time c_e^2 and arrival rate c_a^2 , the number of parallel servers m and the mean process time t_e . Control parameters are given by a specified relative confidence width ϵ and level of precision β . Then the required run length l (in terms of observed customers) is given by Eq.(3.8),

$$l(\epsilon, \beta) \approx \frac{8t_e(c_e^2 + c_a^2)z_{\beta/2}^2}{m\epsilon^2(1-u)^2}, \quad (3.8)$$

where $z_{\beta/2}$ is the normal distribution percentile. Figure 3.6 shows a plot of the minimum required amount of customers to satisfy a G/G/1 queue approximation with a confidence of 95% and relative width $\epsilon = 0.05$, assuming that the system has moderate variability ($c_e^2 = 1$ and $c_a^2 = 1$) and a mean effective process time t_e of 1.

Examining the accompanying values in Table 3.1 for Fig. 3.6, it can be seen that l is

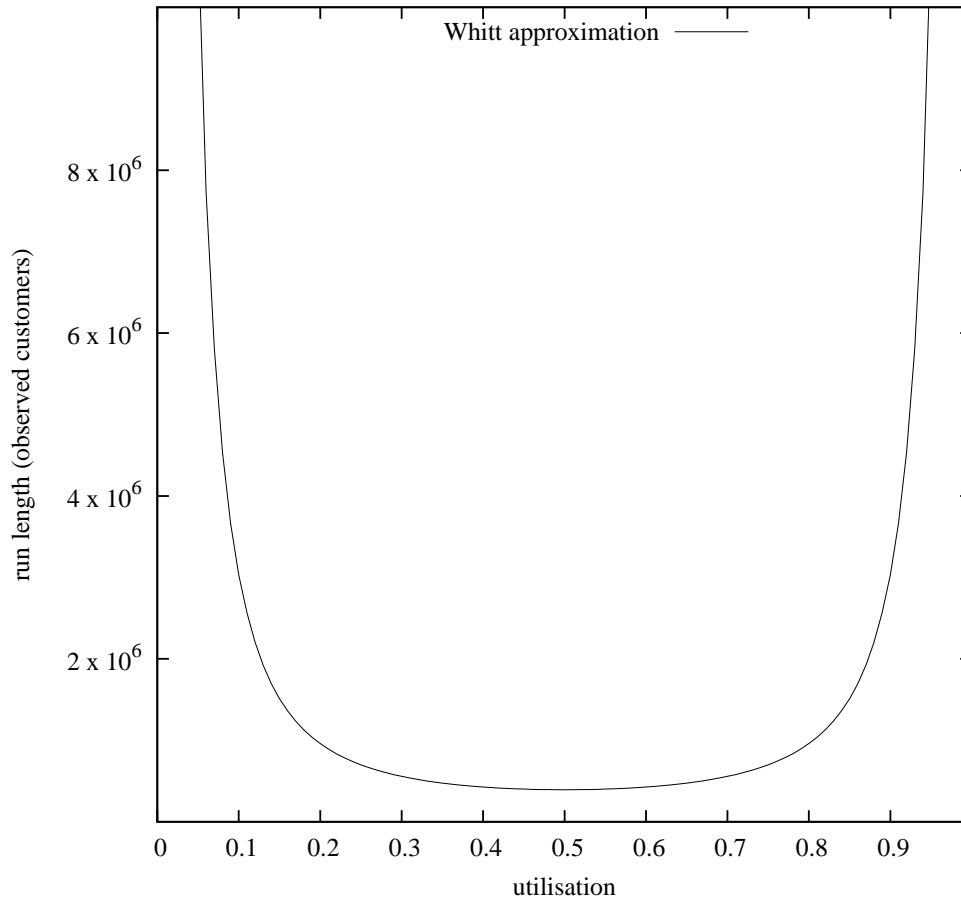


Figure 3.6: Recommended run length for G/G/1 queue with moderate variability.

Table 3.1: Simulation run length approximation for G/G/1 queueing system according to the Whitt estimator (Whitt, 1989b).

u	$l(0.05, 0.95)$ no. of customers
0.01	2.5×10^8
0.1	3.0×10^6
0.2	960364
0.3	557490
0.4	426828
0.5	393365
0.6	426828
0.7	557490
0.8	960364
0.9	3.0×10^6
0.99	2.5×10^8

not strictly increasing for $u \rightarrow 1.0$. Therefore, this equation should only be used in areas of high utilisation. It is not accurate for low or even moderate utilisation levels. This is because it is derived from estimations of variance from fundamental queueing steady state approximations.

Nevertheless, it is incorporated into this framework, as there is value in choosing this method over arbitrarily declaring or guessing an initial run length. It is assumed that any u value less than 0.5 takes on the run length value at $u = 0.5$. Furthermore, any underestimation of run length is nullified by the analysis of the warm up period (see the next section) which ensures that the simulation run length is adequate.

The programmed algorithm of this method is described in Appendix A.2. A relative width of 0.1 was selected as the default. Whitt remarked that if the relative width ϵ is increased tenfold from 0.05 to 0.005 then the run lengths increase almost a hundredfold. Considering that a failsafe of warm up period estimation is employed in the framework, it appears that using relative width of 0.1 as the default value should suffice, given that it is better to have a potentially shorter than required run length and compensate later, rather than to have a very long run that would unnecessarily increase the level of simulation effort.

3.4 Methods for Identifying the Initial Bias and Warm Up Period

The nature of any discrete event simulation (DES) is that it is impossible to start from *steady state*. Steady state describes the portion of the simulation run that is independent from initial starting conditions or initial bias. Similarly, it can be difficult to configure a simulation or ‘load’ it, such that the system is immediately deemed to be in steady state upon starting. Therefore, if using a number of replications there must be some portion of each run, the initial bias period, that must be removed.

Hoad et al. (2008) found 42 methods in total of identifying the initial bias of a simulation run and categorised them under the main headings of graphical, statistical and heuristic procedures. Listing and investigating these methods is somewhat outside the scope of this thesis, however, some authors, including Alexopoulos (2006); Condrón (2010); Gafarian et al. (1978); Mahajan and Ingalls (2004); Robinson (2002, 2005, 2007), have summarised and compared many of the available methods. Many of these authors stated that no one method could be chosen over another in all circumstances.

However, to construct the framework described in this chapter, *it is* necessary to select one method. Statistical methods were selected as the most appropriate for the analyses in this thesis for the following key reasons:

1. Graphical methods, although perhaps the easiest to implement, are extremely subjective to the viewer and cannot be automated without some form of user interaction,
2. Heuristic methods are tractable, simple to comprehend, and easy to implement into programming algorithms. However, many of them are still very much ‘rules of thumb’ which practitioners have devised based on experience. It also appears that there are many caveats and special considerations such as simulation traffic intensity and performance metric selection that affect the usability of these heuristics in broader circumstances (Condrón, 2010),
3. Statistical methods are less efficient in terms of computational time and effort required to conduct the analysis. However, the techniques are based on reliable statistical control methods that are both logical, and have clearly defined algorithms which make them easy to implement and automate. This advantage was seen to outweigh the extra computational effort required.

One particular method, the statistical process control (SPC) method was identified as being particularly appropriate based on its generally applicability.

3.4.1 SPC method

The SPC method described by Robinson (2002, 2007), and implemented by Hoad et al. (2009), uses common process control methods to identify if a time series is in steady state based on the assumption that the warm up period section of the output data is considered to be ‘out of control’ due to variation. The following section describes how this method was implemented. It is based on an algorithm described by Robinson (2002, 2007), with key changes made to how the data is analysed for serial correlation and normality. An outline summary of the method is given as follows:

1. Perform replications and collect data.
2. Determine the appropriate batch means using:
 - the Von Neumann test for serial correlation,
 - the Anderson Darling test for normality.
3. Construct control parameters and identify steady state.

Perform replications and collect data

The length of the simulation run is first calculated using the Whitt approximation, as discussed in Section 3.3, using a black-box fit of an appropriate queueing model for the system. Then five initial replications are performed and the data is collected. To ensure that a sufficient number of replications are performed, the algorithm discussed in Section 3.2 is used to identify the recommended number of runs. If there are $i = 1, 2, 3, \dots, m$ observations collected from each replication. The data is then averaged across the replications, which gives the time series of the average sample means for all replications performed as $\bar{y} = (\bar{Y}_1, \bar{Y}_2, \bar{Y}_3, \dots, \bar{Y}_m)$

Determining the appropriate batch means

The next step involves creating a batch means of the data series to reduce or remove the autocorrelation of the time series. Autocorrelation (also known as serial correlation)

happens as a consequence of monitoring observations in a time series data from simulation output. Any observation collected is somewhat affected by the previous observation. The aim is to reduce this correlation as much as possible. The process of averaging across replications, as in Section 3.4.1, helps to remove some of the serial correlation, and grouping observations into batches aids in further reducing it. This method is known as the batch means method and was popularised by Law and Carson (1979); Law and Kelton (1997).

This method divides the (possibly correlated) time series $\bar{y} = (\bar{Y}_1, \bar{Y}_2, \bar{Y}_3, \dots, \bar{Y}_m)$ into a number of batches h , where the batch size is $k = \lfloor m/h \rfloor$. Then, each batch mean is,

$$\bar{\bar{Y}}_i = \frac{\sum_{i=(h-1)k+1}^{hk} \bar{Y}_i}{k}, \quad (3.9)$$

for $h = 1, 2, \dots, \lfloor m/k \rfloor$, giving the vector $\bar{\bar{y}} = (\bar{\bar{Y}}_1, \bar{\bar{Y}}_2, \bar{\bar{Y}}_3, \dots, \bar{\bar{Y}}_h)$. A coded implementation of this method can be found in Appendix A.5.2.

A technique proposed by Fishman (1978) and used subsequently by Hoad et al. (2009) and Robinson (2002, 2005) is used to locate the appropriate batch size. The initial batch size k is set to 2, the batch means are formed and tested for autocorrelation and normality using the Von Neumann and Anderson-Darling test, respectively. If the data ‘fail’ either test, i.e., data is correlated or not normal, tests that are required to satisfy the assumptions of the SPC method, then the batch size is increased twofold. These tests are carried out repeatedly and any doubling of the batch size is performed until both tests pass, meaning that the data is not correlated and is normal, or the number of batches has reached a critical lower limit ($h > 20$). The next step is then to test the midway point between the ‘failed’ batch size and the ‘passed’ batch size. This process is repeated until the lowest batch size is found that passes both test. A coded implementation of this algorithm is included in Appendix A.3.

Identifying steady state

A time series trend or transient can be considered as ‘in control’ as long as a number of control parameters are not violated. Three sets of control parameters can then be constructed $UL_3, UL_2, UL_1, LL_1, LL_2, LL_3$.

Beginning at the end of the time series and working backwards towards the start, the series is assumed to fail at the first point where any of the following four control limits fail;

Test 1: A point plots below LL_3 or above UL_3 ,

Test 2: Two out of three consecutive points plot below LL_2 or above UL_2 ,

Test 3: Four out of five consecutive points plot below LL_1 or above UL_1 ,

Test 4: Eight consecutive points plot on the same side of the mean μ .

The simulation run length must be at least twice the length of the warm up period, a recommendation given by Kelton (1980). Iterating through each batch mean, if any of the batch means fail in the last half of the series, then it is assumed that steady state was reached in the latter half of the simulation runs and the run length was insufficient. If the test fails in the first half of the series, then the first point where it failed (working backwards) is considered as the point where steady state is assumed to have begun, and any data before this is disregarded.

Another scenario occurs if the time series is ‘in control’ from start to finish, this often happens when the batch sizes are high and much of the initial transient is captured in the first batch. In this case, steady state is assumed to have begun when the time series first crosses the mean line.

An example is shown in Fig. 3.7 for an M/M/1 queue with $t_e = 1$ hour. At a particular design point, the time batched transient \bar{Y}_i of the performance indicator (cycle time) with mean μ is plotted with the control limits. Working backwards at \bar{Y}_{53} the test results are given by Table 3.2. Test 1 fails at \bar{Y}_{36} because this batch mean plots above UL_3 . However, it does not fail at any other batch mean. Similarly none of the other 3 tests fail at any

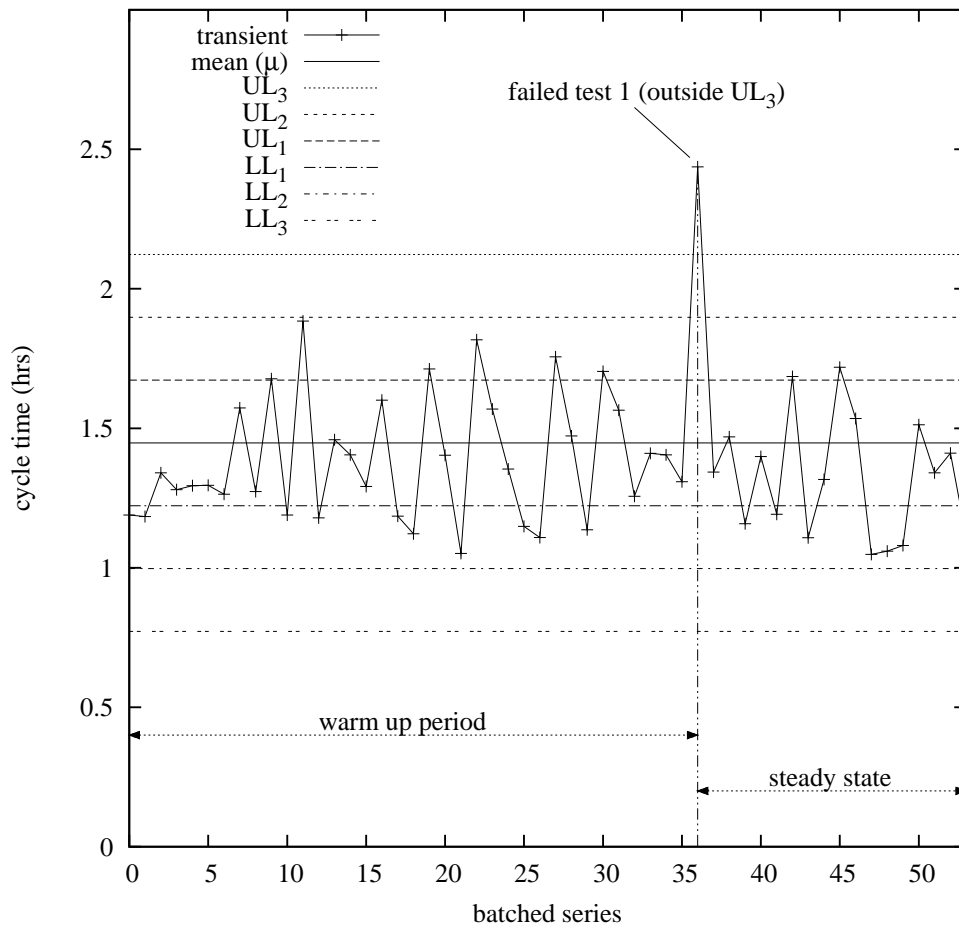


Figure 3.7: Batched time series transient and SPC control parameters for an M/M/1 queue, showing failure of Test 1 at \bar{Y}_{36} .

point along the transient. Therefore, it is assumed that the series is in control after \bar{Y}_{36} , which is in the latter half of the time series (i.e., $\bar{Y}_{36} > \left\lfloor \frac{\bar{Y}_{53}}{2} \right\rfloor = \bar{Y}_{26}$) meaning that the run length is deemed insufficient and the experiment should be rerun with twice the run length.

Table 3.2: SPC test results for M/M/1 queueing system.

SPC Tests	Result
Test 1	Fails at \bar{Y}_{36}
Test 2	Pass
Test 3	Pass
Test 4	Pass

Had Test 1 passed, then all tests would have passed and steady state would be assumed

to be the point where the transient crosses the mean line the first time. In this example, that point would be at \bar{Y}_7 . The code for this algorithm is included in Appendix A.4

3.5 Operational Characteristic Surfaces

All of the operating curves discussed have used 2-D axis plots of either cycle time/x-factor and utilisation/throughput. However, it is also possible to plot a surface of the key relationship between cycle time, utilisation and variability. Operational characteristic surfaces or operating surfaces for short are introduced, to examine these interrelated factors. An operating surface shows the relationship between all three key factors. For example, Fig. 3.8 plots variability on the y-axis for a G/G/1 queue and shows how the resulting x-factor surface ‘peels’ upwards as variability increases.

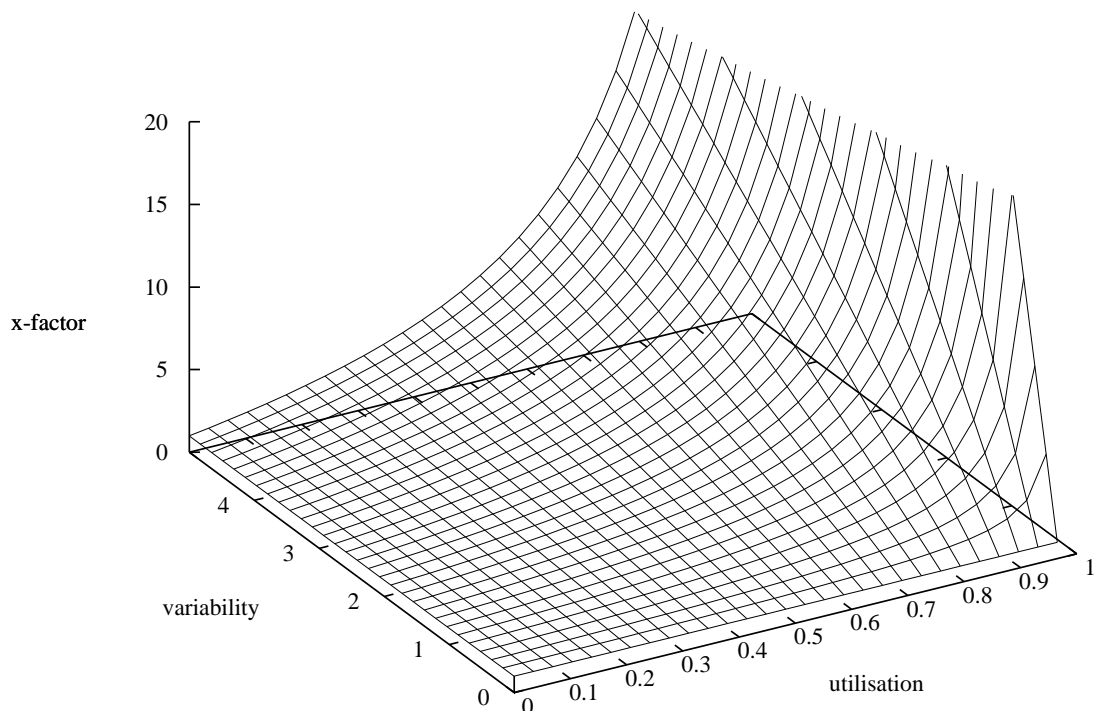


Figure 3.8: Operating Surface for a G/G/1 queueing system according to Eq.(2.4).

This type of surface could also be plotted by fixing the variability and examining an increase in capacity. A similar concept can be applied the u/CT curve, as in Fig. 3.9, which shows that as variability decreases, the u/CT ratio increases towards its maximum

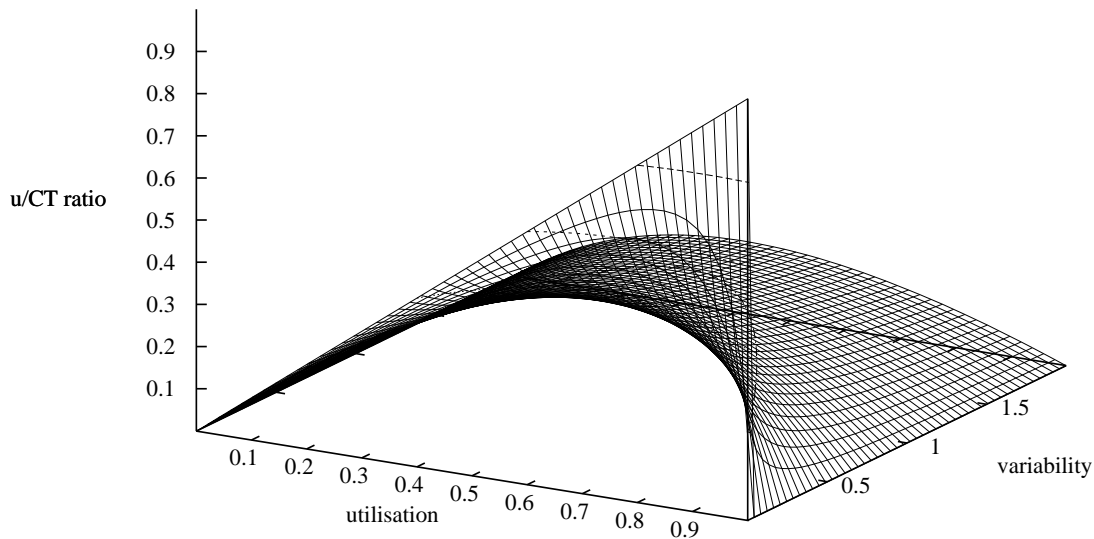


Figure 3.9: u/CT Surface for a G/G/1 queueing system.

allowable envelope and shows the system with the highest efficiency. Whenever possible, these surfaces are used to show how the three components; utilisation, variability, and cycle time interact.

3.6 Summary

This section describes the framework for designing simulation experiments that was automated to speed up the time taken to perform a simulation project. The steps are shown in Fig. 3.10 and the procedure includes:

1. Location of design points on operating curves using equivalent u/CT curves based on queueing theory approximations,
2. Estimation of ‘dry’ run length, i.e., obtaining an estimate of the required simulation run length in the absence of any exploratory data, using an approximation described by Whitt (1989a),
3. An iterative algorithm used to locate the minimum appropriate number of replications given a specified level of precision,
4. Identifying the appropriate minimum batch size for the output data such that the batch means are both normally distributed (according to the Anderson Darling test for normality) and not correlated (according to the Von Neumann test for autocorrelation),

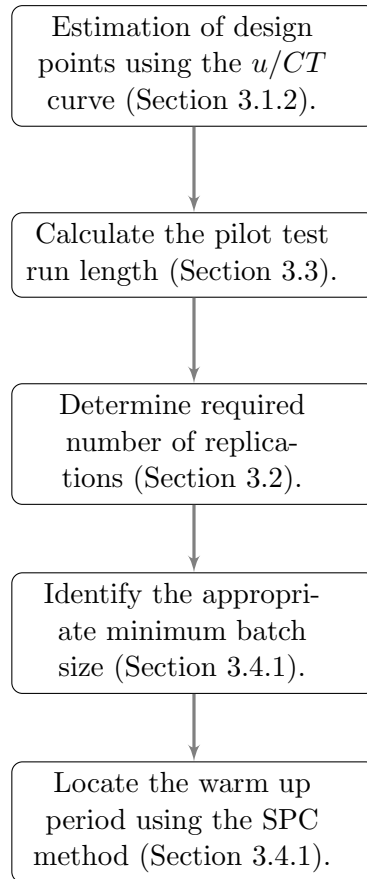


Figure 3.10: Summary flow chart for the automated framework for designing DES experiments.

5. Use of the SPC method for identifying steady state and estimating final required simulation run length.

Key to this framework is the ability to integrate all of the procedural steps, and automate them into a single package into which DES models can be embedded (see Appendix A). It is the aim of the package to speed up the process of obtaining operating curves, and place a focus on analysis and interpretation of the results rather than in the design of experiments.

Case Study: A Flexible Toolset Modelling Application

This chapter describes the complete methodology and creation of a flexible reusable modelling application, known as the Flexible Toolset Modelling (FTM) application, for semiconductor manufacturing toolsets. The aim of the application is to provide the user with an operating curve for any toolset or functional area within the fab in a rapid manner. The application carries out the following procedures:

- User input through a graphical user interface (GUI) style wizard,
- Communicating with local factory databases for identifying, mining and collecting the relevant raw historical data,
- Data filtration, clean-up and outlier screening, in order to correctly interpret the collected data,
- Algorithms for fitting distributions to stochastic patterns,

- Design of experiments (DOE) for simulation using projected theoretical pilot values for a complete simulation analysis,
- Generation of a flexible reusable simulation model in ExtendSim that runs as a background process,
- Direct control and management of simulation variables in ExtendSim,
- Collection of output values from the model,
- Visualisation of simulated operating curves.

ExtendSim was selected as the backbone simulator for the application. ExtendSim is a windows style object-orientated general purpose simulation package that can model a wide range of systems. It has extensive libraries of ‘off the shelf’ blocks that can be dropped into a modelling canvass to represent the entities, events and activities that constitute the real system under investigation. Furthermore, it uses a custom computer language known as *ModL*, through which, users can build their own custom blocks.

4.1 Testbed Background

The case study was conducted in a typical 200mm fabrication facility that produces flash memory and logic, and has a very diverse product and process range, with over 65 individual products. Modelling this environment was a challenge due to this high level of product diversity, however, such a rigorous testbed environment meant that the application had a higher probability of successful deployment at a low product-mix facility.

The aim of the FTM application is to generically model the toolsets in the fab. As a first step, it was necessary to capture and group similar types of tools and processes into discernible categories. While criteria for dividing and grouping the tools could be based on the process make-up or the wafer’s chemical or physical transformation, discrete event simulation (DES) modelling is only concerned with the timing of an event and its order. Therefore, it was decided that the key grouping criteria should be based on the

mechanisms and processes that pass the lots through the tools, meaning, the lot sequence and overlapping of operations is the key criteria. Toolsets are described as;

single process can only process/hold one lot at a time. Therefore, the lot already in process must be unloaded before a subsequent lot can be loaded,

batch allows lots requiring the same specific operation to be batched and processed simultaneously,

cascade lots are overlapped (cascaded) through the tool with no specific operation changeover rules,

batched cascade batched cascade tools are similar to cascade tools, the only difference being that run rules are applied,

cascaded batch batches are formed up to a maximum allowable batch size and then cascaded through the tool.

This type of grouping system facilitated the creation of a generic multifunction flexible model that can isolate the common elements and create a robust model of the toolset or functional area under investigation. The application is not designed to capture the very specifics of a toolset. Rather, it was designed with a ‘point and shoot’ philosophy that offers speed and convenience of use. The hope is that it will be the first application that is used during the decision-making process and before any long term investment of time or resources are committed.

The case study toolset is known here as the ‘H’ toolgroup. It consists of seven tools that fall into the category of ‘single process’, and there are three operations that pass through the toolset. The names of the tools and operations have been changed and all data has been anonymised.

4.2 Front-End for the FTM Application

The front-end of the FTM application has a GUI built using Visual Basic (VB). The accompanying code can be found in Appendix B.1. The SELECT TOOLS tab shown in Fig. 4.1, invites the user to select the tools/toolsets of interest from those available in the

fab using a VB tree structure. Tool selection can be any combination of tools, although, intended usage would involve selection of a particular group of tools in a functional area or toolset. The tools available to the user can be either collected from running a scanning program on the database, or a quicker solution is to upload a list of selectable tools to the FTM program and edit it if tools are made available or redundant on the factory floor.

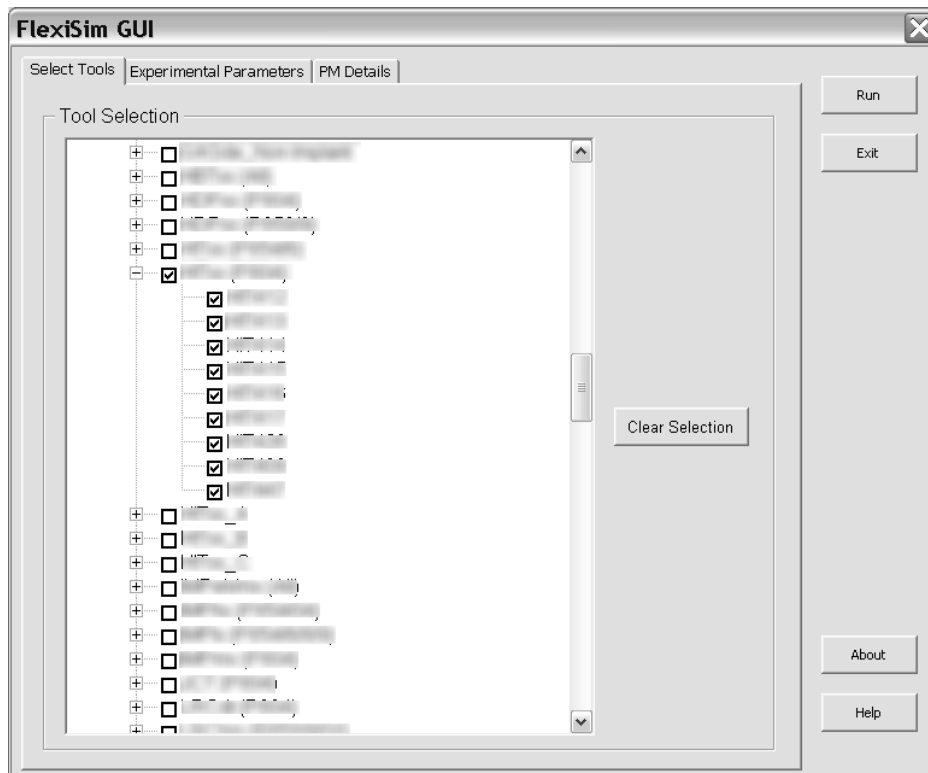


Figure 4.1: Tool/toolset selection for the FTM application.

The user-configurable experimental options are included in the second tab of the GUI as in Fig. 4.2. Here, it is possible to select a historical time window that will be searched back to capture the primary data that drives the simulation model. The user can also select the warm-up period precision in the EXPERIMENTAL PARAMETERS tab, as in Fig. 4.2.

The unscheduled downtime processes, mean time before failure (MTBF) and mean time to repair (MTTR), in the simulation model are constructed from real data, whereas, the preventative maintenance (PM) cycles can either be created by the addition of a number of custom cycles for the tools, or by allowing the program to automatically

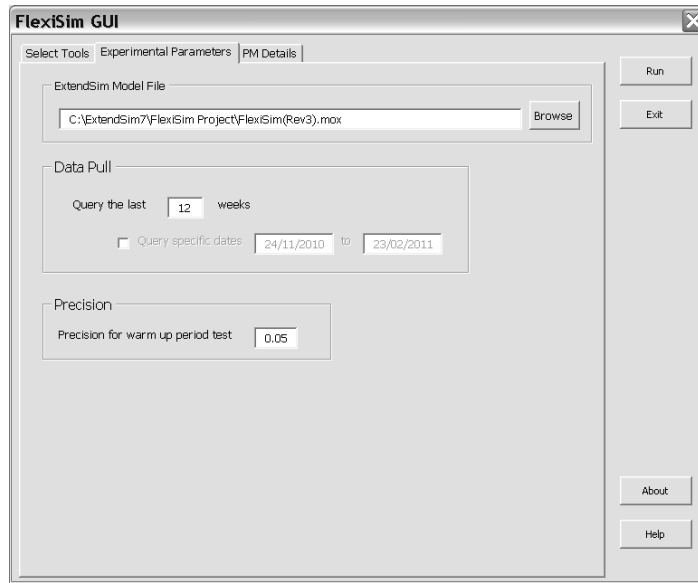


Figure 4.2: Selection of experimental parameters for simulation model.

interpret a PM cycle from the data. The former method is useful for inputting repetitive PM cycles, while the latter method was more useful if there was a lack of knowledge about the tools' PMs.

Many of the tools and equipment within the fab operate PM schedules based on a daily, weekly and monthly period. This method of implementing PM schedules to the model was found to be more stable and more accurate than allowing the program to formulate PM schedules. Addition of the PM cycles is done through an application wizard which consists of a number of steps. The accompany VB code for the GUIs are included in Appendix B.2.

4.3 Data Mining and Collection

There is an extensive amount of data collected in the fab and much of it is stored in an on-site database. This system can be accessed through standard Structured Query Language (SQL) queries. Two tables within this database are of interest to the model; the history of the lots that pass through the selected tools and the history of the tools. These database tables are populated with recordings of events and their corresponding

timestamps. Supplementing this information with another database, known as ‘Tlogs’, gave more detailed information about the actual processing operation on the tools. A cross-reference of events using the captured lot history and tool history in Fig. 4.3, shows the timestamped events and their information sources (Table 4.1). This information was used to reconstruct the process cycle of a tool for the FTM application.

Table 4.1: Time-stamp sources.

Tlogs	Lot History	Tool History
Intro	PREV_OUTDATE	BEGIN RUN
ExecA	IN_DATE	END PROCESS
Done	OUT_DATE	END RUN
MoveOut		

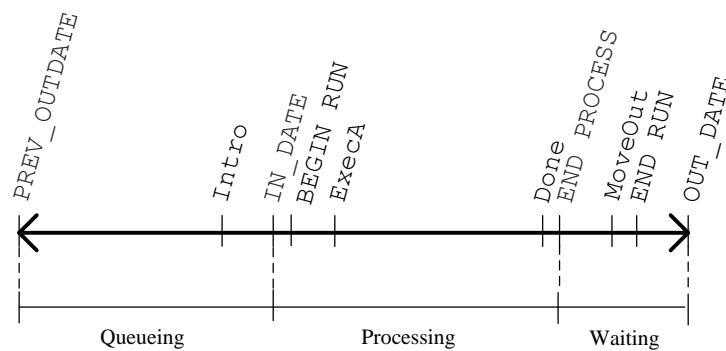


Figure 4.3: Combined database time stamps for a single process tool.

The ‘lot history’ database records the movement of lots through a tool as well as the lot ID and the operation ID. A single line entry corresponds with the time that a lot completes its previous operation and departs from the previous toolset. At that point ‘ownership’ of the lot is transferred to next operation and toolset (denoted by the timestamp PREVOUT_DATE), meaning that technically the lot is considered to be queueing for the destination tool even though it may still be travelling towards that tool. The timestamp IN_DATE is considered the point where the lot has been assigned to processing on the current tool, and the lot history timestamp OUT_DATE is the point where ownership is transferred to the downstream tool and the lot is considered to have left the area.

The ‘tool history’ records events from a tool perspective. A single event recording includes the time of the event, the tool ID, the current availability status indicator (whether it is up or down), the new availability status indicator, and the actual status of the tool. The status of the tool indicates whether it is loading, processing, in PM, or in repair, etc. There is no lot identification recorded in the tool history table so the data needs to be cross-referenced with the lot history to match up the lots that passed through the tool. The tool history table is also the primary source of all downtime information regarding the tool.

4.3.1 Determining arrival patterns

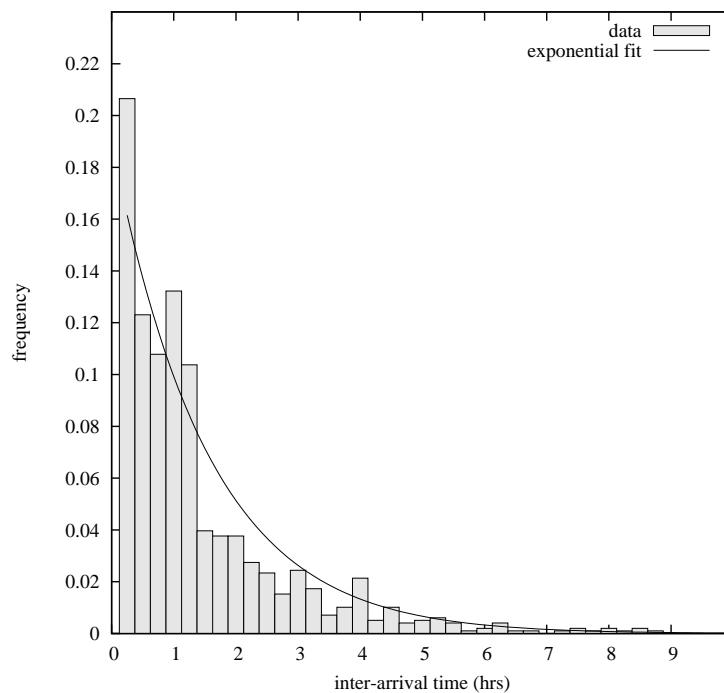


Figure 4.4: Arrival histogram and exponential fit for lots requiring operation A on ‘H’ toolset.

A number of operations are performed on the data. One of these is the determination of the arrival pattern distribution of the lots to the toolset. The inter-arrival rate was measured as the time between successive arrivals denoted by the PREV_OUTDATE in the lot’s timestamp history. A coded implementation of this algorithm can be found in Appendix B.2.2. It was found that by categorising the arriving lots by their required

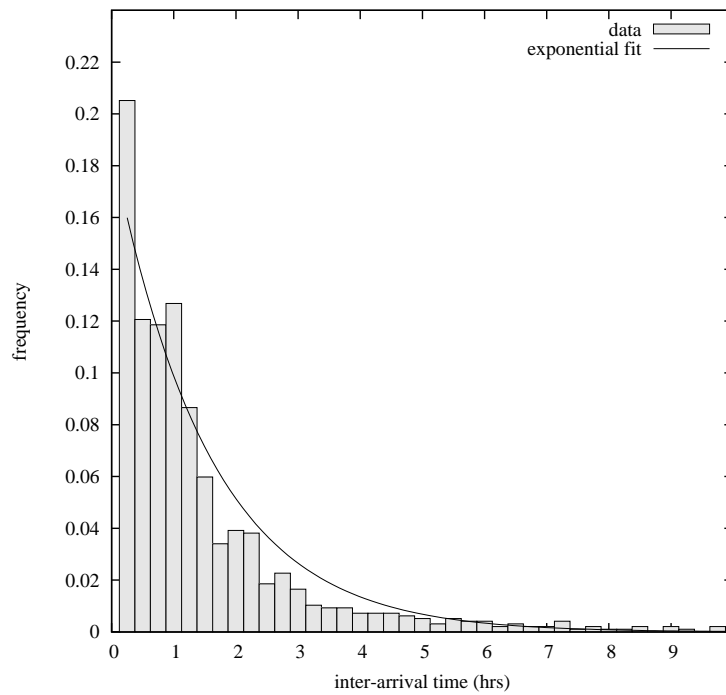


Figure 4.5: Arrival histogram and exponential fit for lots requiring operation B on 'H' toolset.

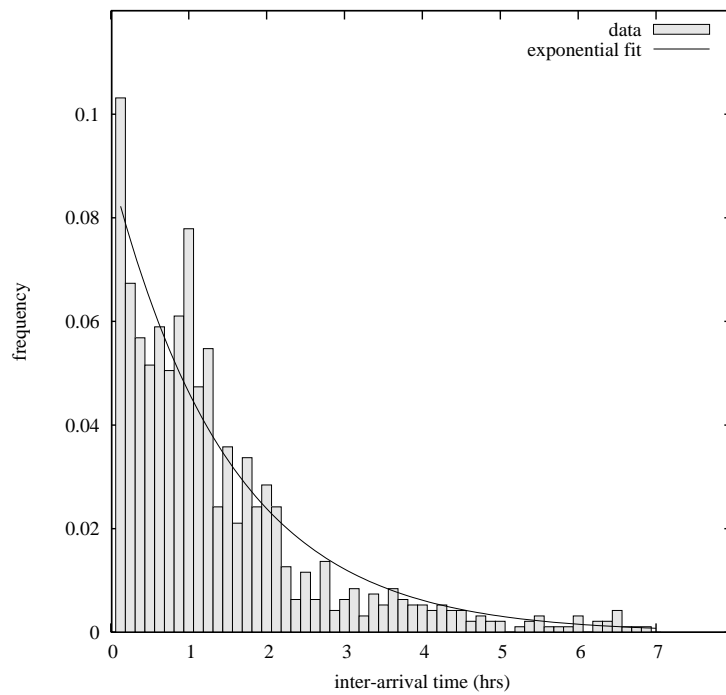


Figure 4.6: Arrival histogram and exponential fit for lots requiring operation C on 'H' toolset.

operation type, a stochastic pattern could be found for each of the lot operation types and that typically this stochastic mechanism could be closely approximated by an exponential distribution, as shown in Figs. 4.4-4.6.

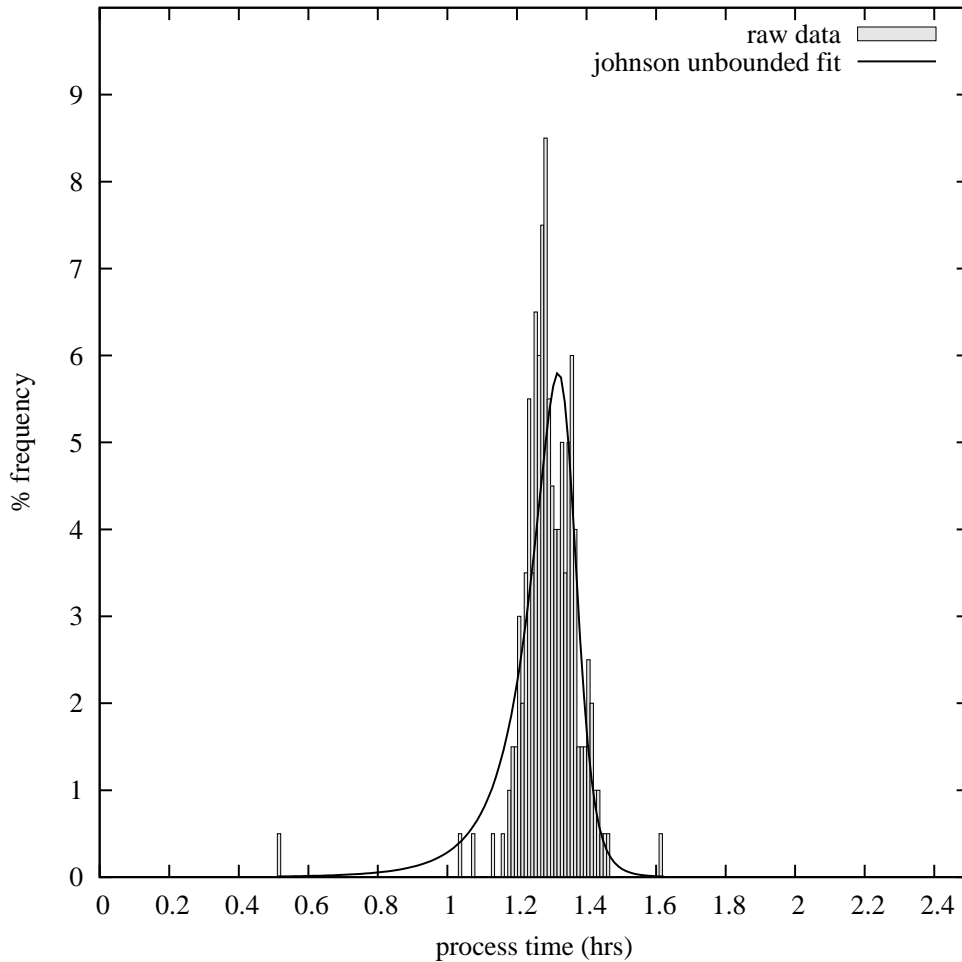


Figure 4.7: Unbounded Johnson distribution fit for lots requiring operation A on tool T2.

4.3.2 Determining lot processing patterns

The lot process duration for the model is found by assuming that the tool begins processing a lot at the `Intro` timestamp recorded in the ‘Tlogs’ and finishes at the `END_PROCESS` timestamp in the tool history database. The distribution for each operation type that passes through the toolset is then fitted to a Johnson distribution using an algorithm described in Appendix F. Appendix F also includes a discussion on the justifications for

using the Johnson distribution for uni-modal processing time data. A coded implementation of the Johnson distribution fitting program (Appendix F.4) returns four parameters that can be used by the simulation model to reconstruct the processing pattern on the tool. The four parameters are; the starting location ϵ , the range λ , the skewness γ and a shape factor η . Also returned, is the Johnson distribution type; bounded (S_B) or unbounded (S_U). Table 4.2 show the Johnson parameters derived from this method for each operation type on each tool.

Table 4.2: Johnson distribution parameters for each operation on each tool in toolset ‘H’.

operation	tool	type	ϵ	λ	γ	η	no.
A	T1	S_B	1.23	0.27	-0.82	1.71	244
	T2	S_U	1.86	0.07	0.84	1.39	250
	T3	S_B	1.27	0.26	0.35	0.84	170
	T4	S_U	1.12	0	1.54	0.86	148
	T5	S_U	1.18	0.01	2.29	1.7	135
	T6	S_U	1.17	0	1.46	0.71	106
	T7	S_U	1.21	0.01	2.37	1.9	127
B	T1	S_U	1.45	0.01	0	0.76	304
	T2	S_U	1.46	0.02	1.93	1.12	90
	T3	n/a	n/a	n/a	n/a	n/a	0
	T4	S_U	1.31	0	1.45	0.78	258
	T5	S_U	1.37	0	-0.27	0.68	226
	T6	S_B	0.99	0.36	-5	1.69	156
	T7	S_U	1.39	0	-0.37	0.84	81
C	T1	S_U	1.58	0.01	1.44	1.87	63
	T2	S_U	1.6	0.01	2.09	2.01	266
	T3	S_U	1.66	0	2.04	1.04	164
	T4	S_U	1.61	0	3.56	0.98	228
	T5	S_U	1.49	0	-0.1	0.53	61
	T6	S_U	1.47	0.01	0.99	1.16	134
	T7	S_U	1.6	0.09	-0.06	1.53	224

Once the lots have finished processing they are removed from the tool, scanned out, and placed back into the stocker. This is not always done immediately, and therefore, there is an additional time delay where the lot is still ‘owned’ by the tool. Samples of this delay were recorded by comparing the END PROCESS time in the tool history to the OUT_DATE timestamp in the lot history. Samples of these delays were collected for each tool and an exponential distribution was fitted to the data for use by the simulation

model. Therefore, the only parameter required for the post-processing waiting time was the mean. The mean values found are shown in Table 4.3.

Table 4.3: Average post-processing waiting time of all lots on each tool.

tool	ave. move out time (hrs)
T1	0.0663
T2	0.0848
T3	0.1133
T4	0.0757
T5	0.1645
T6	0.0277
T7	0.1449

4.3.3 Downtime event distributions

Most tools in the fab suffer from unexpected outages or unscheduled down events. They are also subject to a number of PM procedures loosely based around daily, weekly and monthly schedules. Capturing this information from the database required investigation of the tool history database.

The tool is assumed to be down, i.e., unable to process lots, when the availability status indicator in the database is checked with a ‘D’. The tool is considered to be back online when the ‘D’ check mark is gone. The program records the corresponding tag for all of these ‘down events’ and presents the user with an option of selecting which tags refer to an unscheduled event and which refer to a scheduled PM event, similar to Fig. 4.8.

Distinguishing between a regular PM and unscheduled can be further complicated by the flexibility of PM schedules. Quite often a PM can be brought forward if the tool begins to show signs of falling outside of normal operational control parameters. The question then becomes, whether the resulting downtime event should be considered scheduled or unscheduled. In this situation it was assumed that if a PM was performed after there were any OUT OF CONTROL error messages then the down event was considered to be an unscheduled downtime, and not a PM.

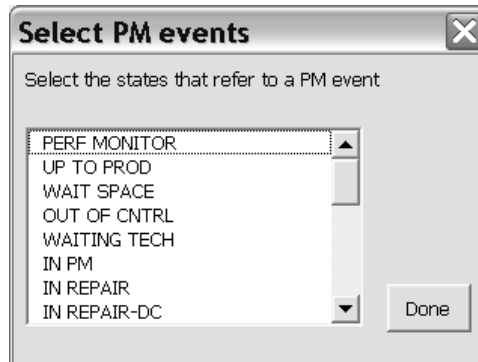


Figure 4.8: User prompt to distinguish between scheduled and unscheduled downtime events recorded in the tool history database.

Another problem when classifying the true downtime nature of the tools, is the intermittent downtime and offline reporting when a tool is in repair. Quite often, the tool can report temporarily that it is back online only to be taken offline again due to some other offline event. To overcome this, it was decided that a minimum time should exist between down events for them to be individual and unique events. Any downtime events within this minimum time horizon were considered to be a result of the same ‘failure’. Recommendations from process engineers in the fab recommended that this minimum timeframe should be approximately 1 hour.

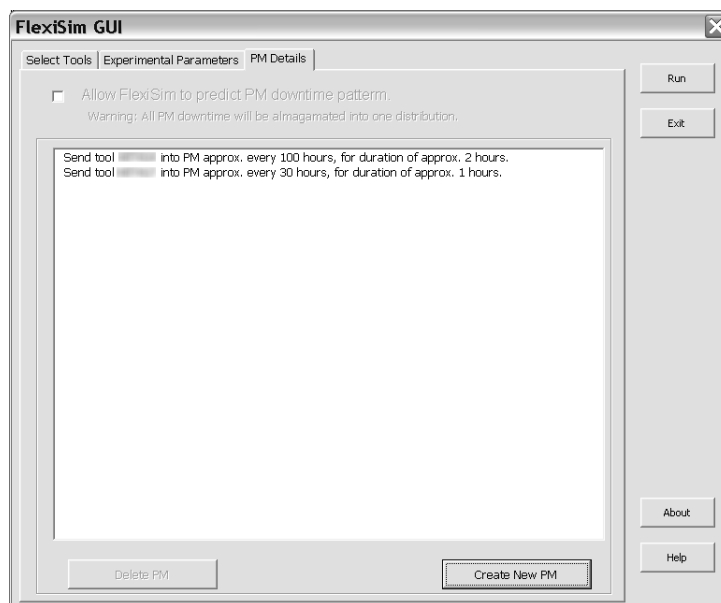


Figure 4.9: Creation of PM schedules through a GUI wizard.

Once the scheduled and unscheduled downtime has been separated and collected, the

program attempts to classify the unscheduled downtime using an exponential distribution, and the scheduled downtime using a Johnson distribution. Due to the flexibility of the PM schedules the program allows the addition of custom PM schedules rather than collecting them from the data. The custom PM entry method allows the users to apply their local knowledge of the tools' PM cycles and add a number of parameters for each tool, as can be seen from Fig. 4.9. The accompanying code for this process is given in Appendix B.2.4.

4.3.4 Lot selection and prioritisation of operations

Most of the lot selection and lot-processing priority decisions in the fab are controlled by a custom computer program. The program is quite complex and therefore, its logic is not used or implemented in the case study. For the purposes of general applicability of the FTM application, it was decided the priority and scheduling of the model should be based on the following four ranking options:

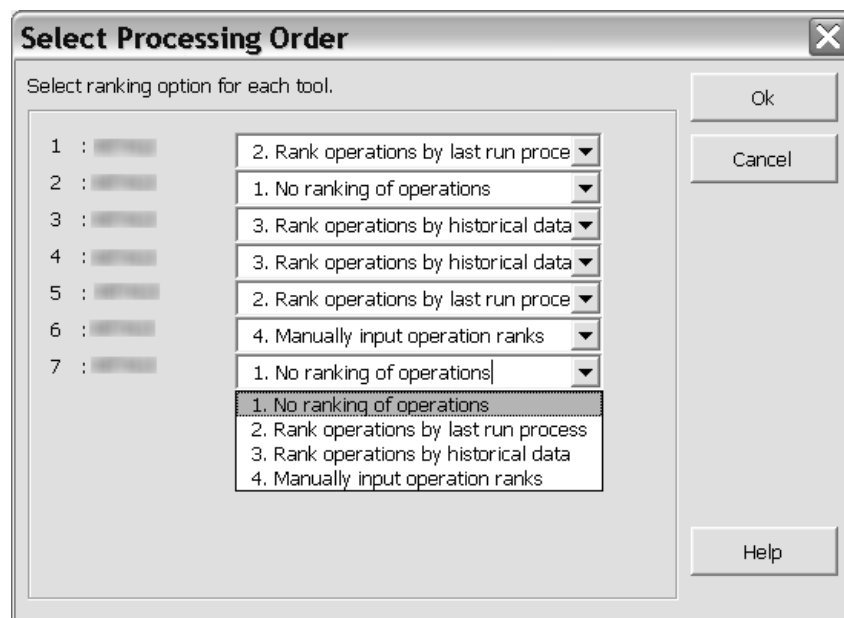


Figure 4.10: VB Userform used to select lot prioritisation options for each tool.

1. No ranking of operations,
2. Rank operations by last run process,
3. Rank operations by historical data,

4. Manually input operation ranks.

The first option means that tools are free to choose any lot and will default to a first in first out (FIFO) system. The second option ranks higher the lots that have been processed last by the tool. This is a typical prioritisation method used when attempting to minimise setups. The tool adopts a FIFO strategy for any tie-breaking situations.

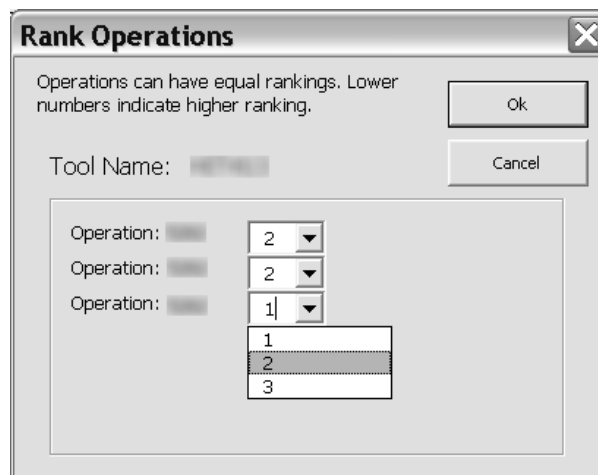


Figure 4.11: VB Userform used to rank processing priority for operations.

The third option provided to the user is to rank the operation types by using the information stored in the database. This means that if a tool processed more of a particular operation type during the period of investigation, then that operation type will have a higher priority on that tool. The final option is to allow the user to custom rank the operations, as in Fig. 4.11. A user can give any permutation of ranking, including equal ranks to two or more operation types. The accompanying code for these operation-ranking wizards is given in Appendix B.2.5.

4.3.5 Exporting information to ExtendSim

The FTM program loads the distributional information and other simulation parameters to ExtendSim. ExtendSim works efficiently with its own internal global arrays and passing the model parameters into these global arrays proved to be an effective solution to communicate between ExtendSim and the VB program. Table 4.4 shows the information

passed to ExtendSim's global arrays. It can be seen that a minimal amount of information is required by the simulation model to reconstruct the stochastic distributions used to model the real system.

Table 4.4: Information required by ExtendSim simulation model.

Name	Dimensions	Type	Description
gaArrivalInfo	(r,3)	real	Lot inter-arrival distribution. <ol style="list-style-type: none"> 1. Operation ID 2. Mean inter-arrival time 3. Percentage operation mix
gaTool	(r,10)	real	Process time distribution info. <ol style="list-style-type: none"> 1. Tool ID 2. Tool type 3. Operation ID 4. Process time distribution type:- <ol style="list-style-type: none"> 1: bounded 2: unbounded 5. Johnson η parameter 6. Johnson γ parameter 7. Johnson λ parameter 8. Johnson ϵ parameter 9. Mean move-out time 10. Percentage of all unique operation and tool combinations
gaPM	(r,3)	real	Preventative maintenance distribution parameters. <ol style="list-style-type: none"> 1. Tool ID 2. Mean uptime 3. Mean downtime
gaDT	(r,3)	real	Unscheduled downtime distribution parameters. <ol style="list-style-type: none"> 1. Tool ID 2. Mean uptime 3. Mean downtime
gaOpRank_xxx	(r,3)	real	The suffix no. xxx refers to the tool ID. <ol style="list-style-type: none"> 1. Operation ID 2. Ranking <ol style="list-style-type: none"> (a) -1: (priority to last processed) (b) 0: (no priorities) (c) 1,2,3,... (lowest no. is highest priority)

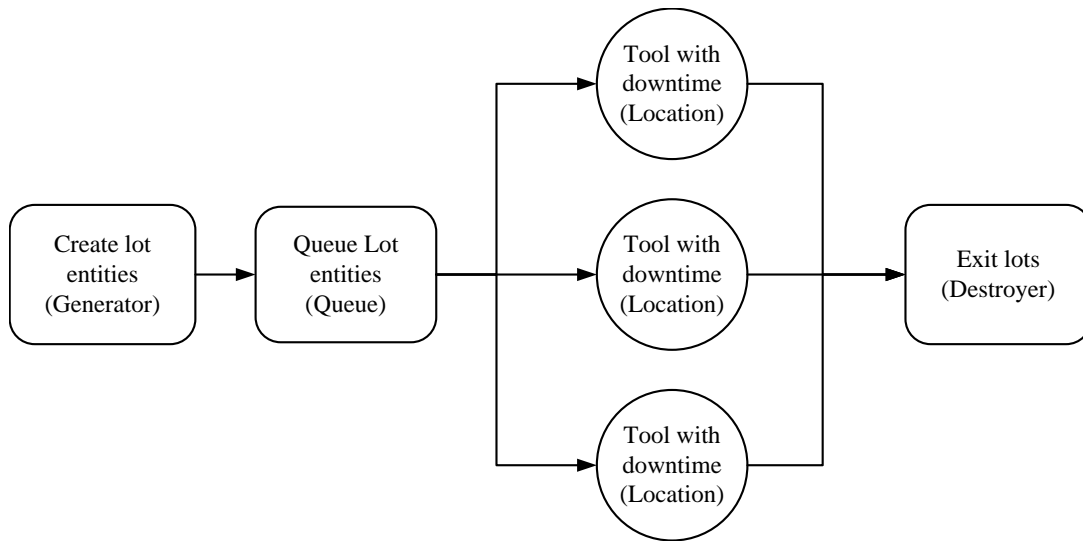


Figure 4.12: Traditional job-driven graphical modelling approach.

4.4 ExtendSim DES Model

As discussed in Section 2.5.4, it can be difficult to implement a flexible reusable simulation model using a graphical simulation package, like ExtendSim. For example, objects like machines are treated as static blocks, while entities such as lots and batches pass through these static objects (as in Fig. 4.12). This traditional modelling strategy is mainly due to the job-driven paradigm that so many general purpose graphical simulators support. In these simulators, a ‘map’ of the system is laid out using static modelling blocks to represent fixed machinery, while the job (a moveable entity) passes through these static resource blocks (Fowler and Rose, 2004).

This poses a problem when the number of tools and entities in the system are many, or the number of tools is variable. Hence any change to the system specification would require physically adding and/or removing blocks to or from the model. In order to circumvent this, it was decided to treat tools and downtimes as entities that circulate a system of processes (modelled using static blocks). This way, it was easier to control aspects of the system such as the number of tools, the tool attributes and the tool downtime or PM events. This also meant that only the minimum amount of blocks were required to model the system (approximately 30), as can be seen from Fig. 4.13,

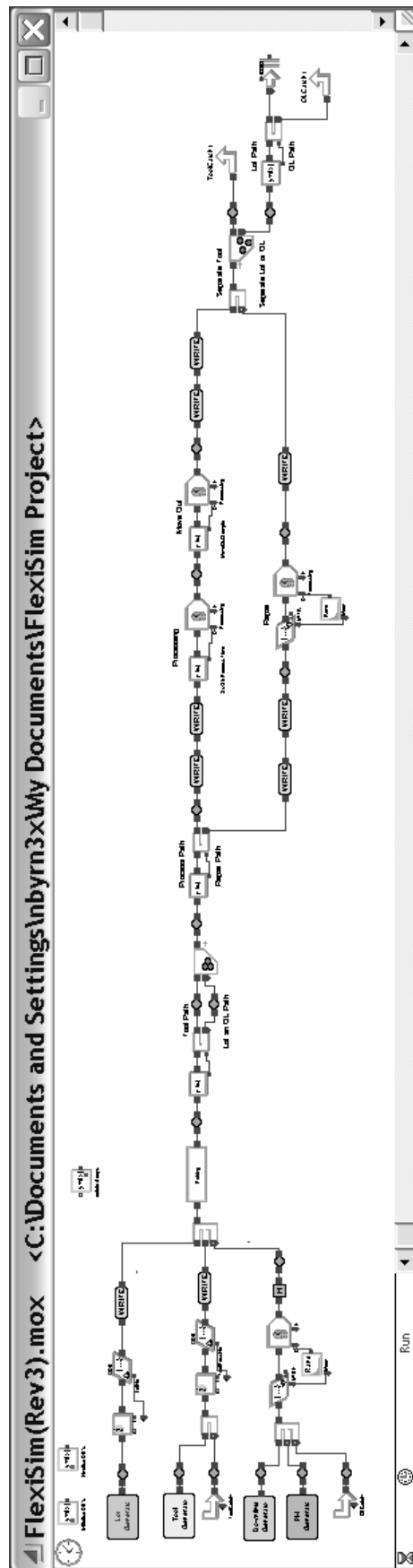


Figure 4.13: Screenshot of ExtendSim model used by the FTM application.

which drastically reduced the amount of code ExtendSim had to process, and significantly reduced the model execution run time.

4.4.1 Lot Generator block

The stock *Create* block in ExtendSim is useful for modelling lot arrivals with a singular stochastic pattern or from a single distribution type, however, for the model required here it is insufficient when there are a number of different lot types (distinguished by the operation type). As an alternative, a multi-lot generator block was custom built using the ModL language in ExtendSim.

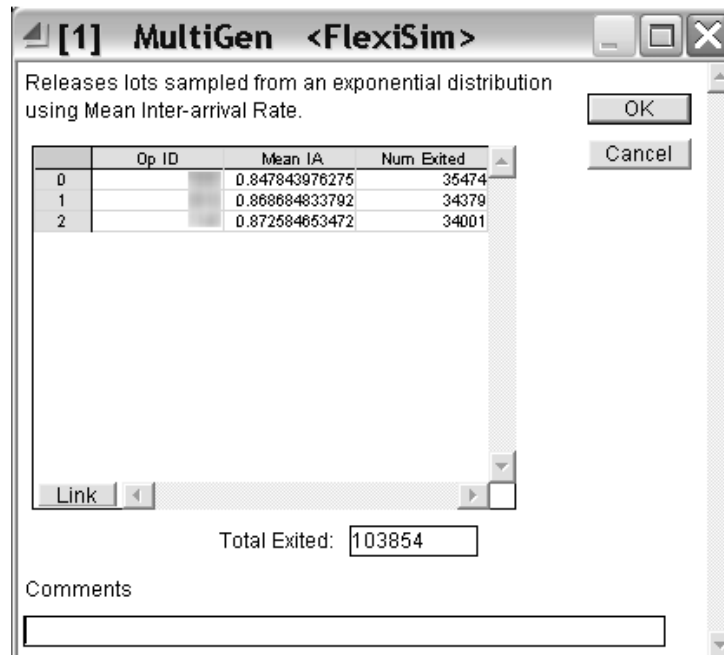


Figure 4.14: Dialog of the custom Lot Generator block used for the FTM application.

This block reads all the lot inter-arrival distribution information that was passed in from the VB application. The block then reconstructs the distributions, samples from them, and creates a lot entity as required during runtime. Using this method, any number of inter-arrival patterns can be modelled, and thus, only one block is needed instead of using an individual block for each arrival pattern. The Lot Generator block also attaches the necessary attribute information to the outgoing lots including; the wafer quantity

of the lot, the operation type and item type. The code for this block can be found in Appendix B.4.1.

4.4.2 Tool Generator block

The Tool Generator block (Fig. 4.15) reads the *gaTool* global array that was populated by the VB program. At the start of the simulation, the block creates a tool item for each tool in the array and attaches attribute information such as tool name, tool type, allowable operations, processing distribution information for each operation and the mean move-out time for that tool (see Table 4.5). The tool items are then released into the model. This method ensures that any number of tools or any tool configuration type can be loaded into the model extremely quickly and easily without having to directly or physically alter anything in the graphical model. The ModL code for this block can be found in Appendix B.4.2.

Table 4.5: Attributes used by ExtendSim model items.

Attribute Name	Description
opID	The name of the operation
itemType	The item type: <ol style="list-style-type: none"> 1. Lot 2. Tool 3. Downtime 4. Preventative Maintenance
toolID	The tool type: <ol style="list-style-type: none"> 1. Single process 2. batch 3. Cascade 4. Batched cascade 5. Cascaded batch
MTBF	Stores the mean time before failure
MTTR	Stores the mean repair time
entryTime	Timestamp of when the item entered the model

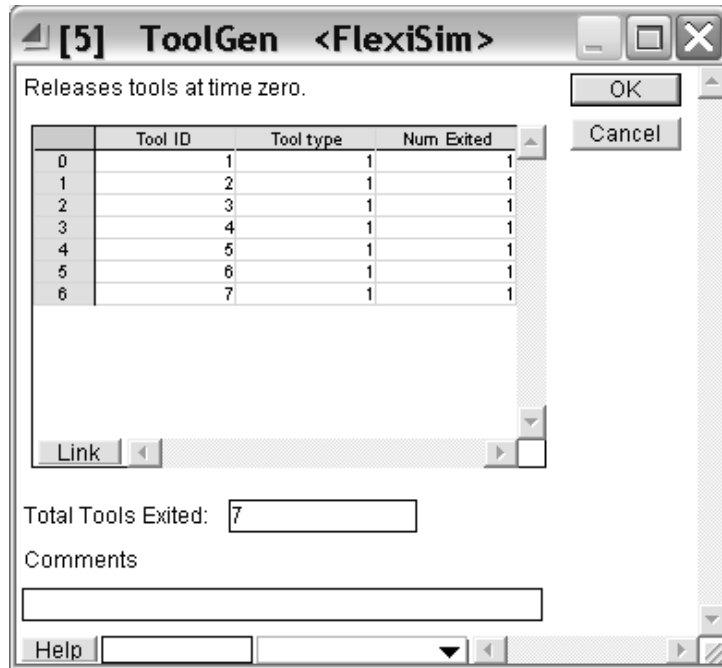


Figure 4.15: Dialog of the custom Tool Generator block used in the FTM application.

4.4.3 Unscheduled downtime generator block

Traditionally most graphical simulation software packages allow the user to input downtime and repair characteristics to machines or processes that are modelled using a single block. However, for the flexible model described here, with tools modelled as items circulating the system, a workaround was required to accurately model downtime and repair patterns. Therefore, it was decided that downtime events should be modelled as items similar to how tools were modelled. The unscheduled downtime items are created by a custom block (Fig. 4.16) which assigns attribute information and releases the items into the model. Once a tool is due to go into an unscheduled downtime event the downtime item searches for its matching tool, prevents it from joining with a lot and restricts it from any other activity. The ModL code for this block can be found in Appendix B.4.3.

4.4.4 PM Generator block

The PM Generator block (Fig. 4.17) is similar to the unscheduled downtime generator block, both in terms of its function and design. The main difference is that it creates

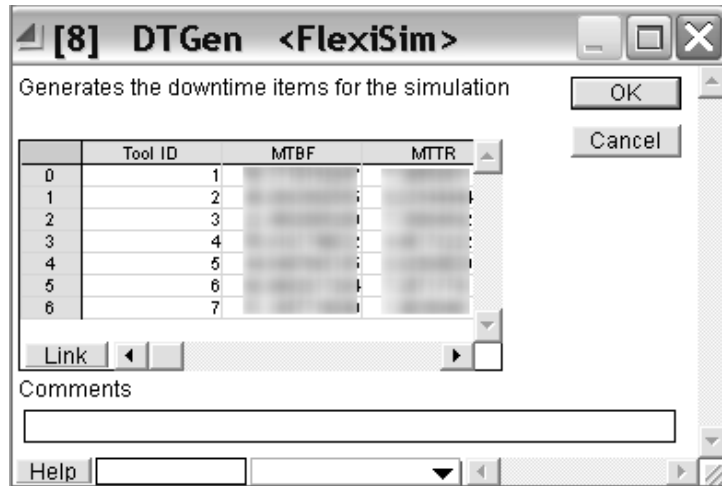


Figure 4.16: Dialog of the custom unscheduled downtime generator block used in ExtendSim.

multiple PM items for each tool to represent the various maintenance cycles experienced by the tools. The ModL code for this block can be found in Appendix B.4.4.

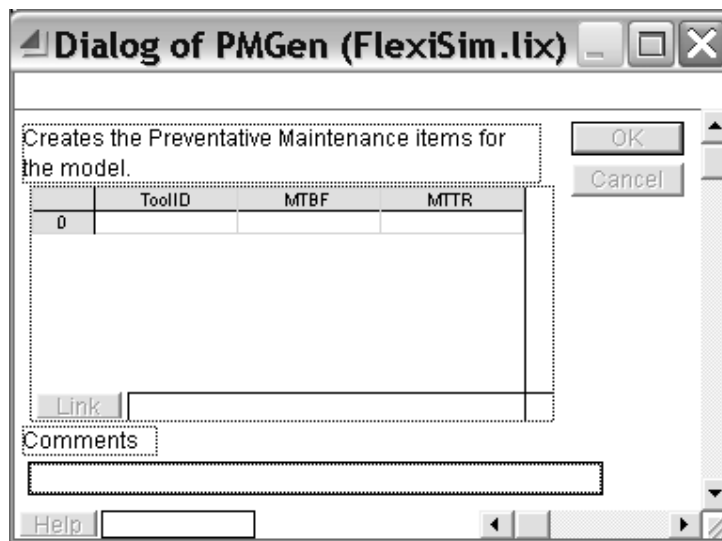


Figure 4.17: Dialog of the custom PM Generator block used in ExtendSim.

4.4.5 Pairing block

The Pairing block is responsible for storing any of the necessary tool, lot, PM or downtime items to be paired, and releasing them to signify the occurrence of some event. For example, the pairing of a lot and a tool would signify a lot being processed on a tool. The pairing of a tool and a downtime item signifies the tool going offline for unscheduled

repair. The Pairing block holds all these items when they are dormant. For example, if a lot is waiting in the block then it is waiting for a tool and none is available. If a tool is residing in the block, then it is considered idle and available with nothing to process. If either a PM or unscheduled downtime item is residing in the Pairing block, then it is awaiting return of its matching tool to pair with, which signifies that the tool has gone offline.

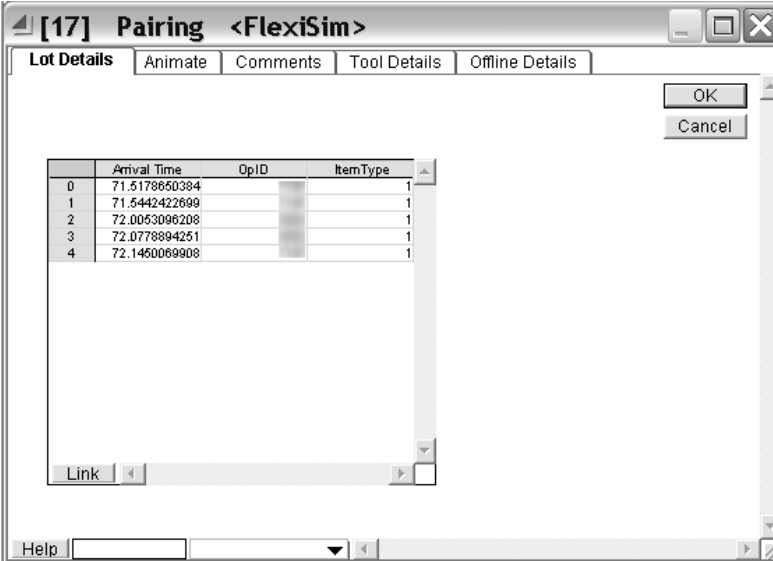


Figure 4.18: Logic code execution for Pairing block used in the FTM application.

The concept of a tool waiting to go into an offline state, as a result of an unscheduled downtime, seems contrary to the definition of unscheduled downtime. Nevertheless, this is a consequence of modelling tools and downtime as items, as opposed to modelling them using the traditional method. If a tool is due for an unscheduled downtime event, it must wait until it has finished what it is doing first, which could be processing a lot or currently

in repair. This means that there will typically be a delay between the simulation MTBF and the actual MTBF. No investigation was conducted into the extent of this difference, however, the difference should be minimal if the nominal MTBF is far larger than a typical PM repair time or lot process time. The impact would also be minimal if the waiting tool had a low utilisation.

There are two main procedures used in the Pairing block; `CheckOffline` and `CheckLots`. The conditions required to run each are shown in Fig. 4.18. The `CheckLots` procedure works by searching for allowable operations for each tool in the Pairing block's internal tool item list and selecting one (if available) based on the ranking assigned by the user (see Section 4.3.4). The `CheckOffline` procedure checks the block's internal tool list and its offline item list for any matching tool names. If any are found, the tool and the offline item are paired and released from the block, signifying that the tool has gone offline. The ModL code for the custom Pairing block is given in Appendix B.4.5.



	Arrival Time	OpID	ItemType
0	71.5178650384		1
1	71.5442422699		1
2	72.0053096208		1
3	72.0778894251		1
4	72.1450069908		1

Figure 4.19: Details of lots residing in the Pairing block during runtime.

The Pairing block's dialog has dynamic tables which update a list of the items stored in the block, which was helpful for debugging. Figure 4.19 shows the LOT DETAILS tab which holds information about lots that are currently in the block and waiting for a tool to pair with, and be processed.

4.4.6 Activity delay paths

Once items have been paired and released (Fig. 4.20), they are sent to activity delay paths to represent the required operation or event. A lot and tool pairing needs to be delayed to represent processing and unloading. This is done by two separate ExtendSim *Activity* blocks which hold the items for a specified period of time. The first delay block, used to represent processing, samples from a Johnson distribution for the Johnson parameters attached to the lot. The second delay block, the Move Out block, is used to represent the time taken for an operator or machine to place the lot back into the stocker. It samples an exponential distribution taken from the mean move-out time parameter attached to the lot. These Activity blocks are stock from the ExtendSim libraries and can hold any number of items. During the simulation, all processing lots can be found in either of these blocks at any given time. After the lot and tool have finished processing and unloading they are unpaired, the lot leaves the system and the tool re-enters the Pairing block again.

Similarly, a tool/offline item pairing will be sent through a delay path to represent the repair of the tool or a maintenance task. The repair time is calculated by sampling from an exponential distribution with a mean based on the MTTR parameter attached as an attribute to the tool. Any tools in the repair Activity block are considered offline. Similarly, any offline items waiting in the MTBF Activity block before entering the Pairing block signifies that its corresponding tool is online. After the repair/maintenance event, the tool and offline item are unpaired, the tool returns to the Pairing block, while the offline item returns to the MTBF Activity block where it is held until the next MTBF time expires

4.5 Recording Simulation Data from ExtendSim

In a traditional modelling setup, it is possible to collect summary information directly from the stock blocks as they carry statistics for most common metrics. However, because the flexible modelling strategy implemented here uses standard blocks in an unorthodox

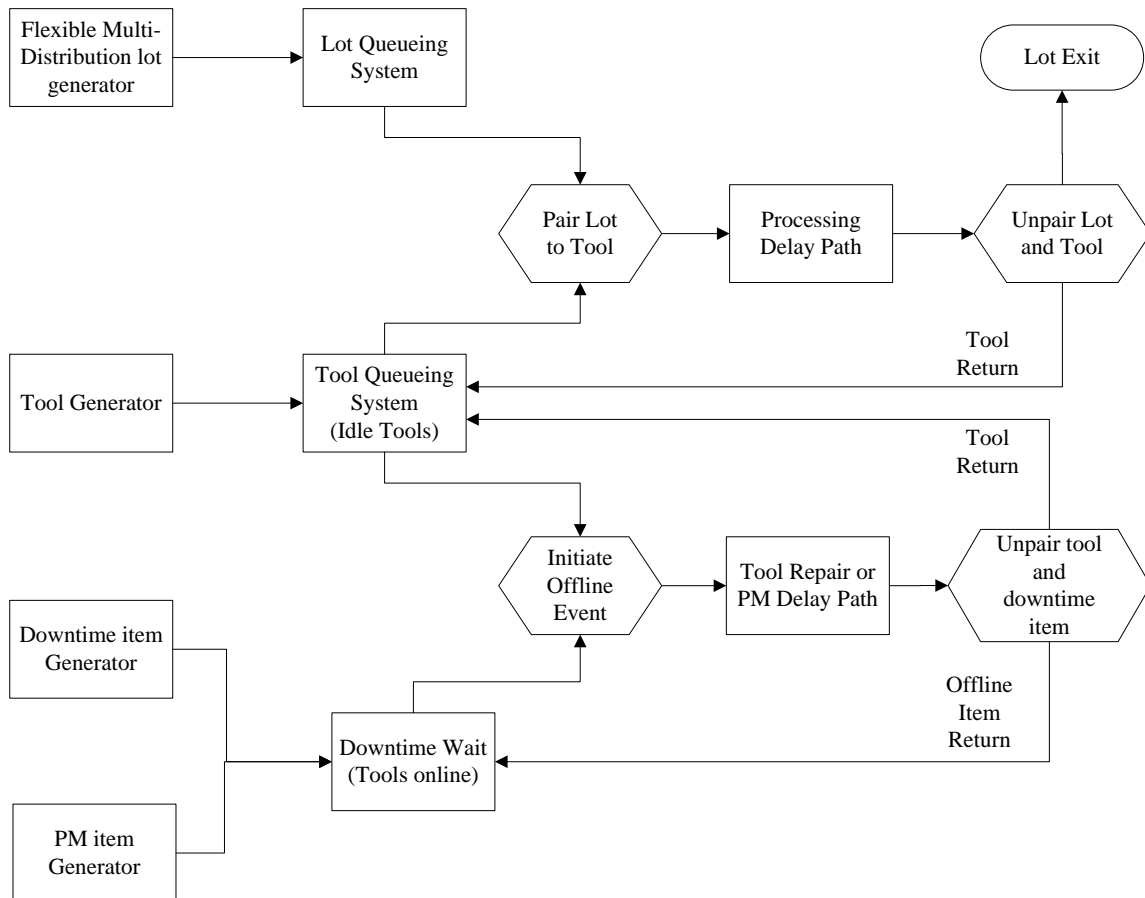


Figure 4.20: Flow system for flexible simulation model used in the FTM application.

way, the statistics collected are invalid. Therefore, to collect statistics, it was decided to record timestamp information in the model, similar to how it is recorded in the actual fab. At various points through the model, specific event information is recorded, as in Table 4.6.

Table 4.6: Model time stamps recorded during runtime.

lot	tool	offline
start time	start time	start offline time
start process	start process time	end offline time
end processing	end process time	
	start offline time	
	end offline time	

This information is recorded in the ExtendSim database. Two tables are used; *lotTrace* and *toolTrace*, that hold the timestamps for the lots and the tools, respectively. Once the

model has finished executing, control is returned to the VB program and it imports the ExtendSim database using functions included in Appendix B.5.

4.6 Generating the Operating Curve

The cycle time and utilisation of the system are calculated for each design point to capture a full map of the operating curve for the system. Cycle time is calculated by collecting the average time spent in the simulation model for all the lot items. Utilisation at each design point is calculated from the tool timestamps and an estimate of the workstation utilisation is made by averaging each tool's utilisation. The operating curve is displayed to the user in an embedded window in VB.

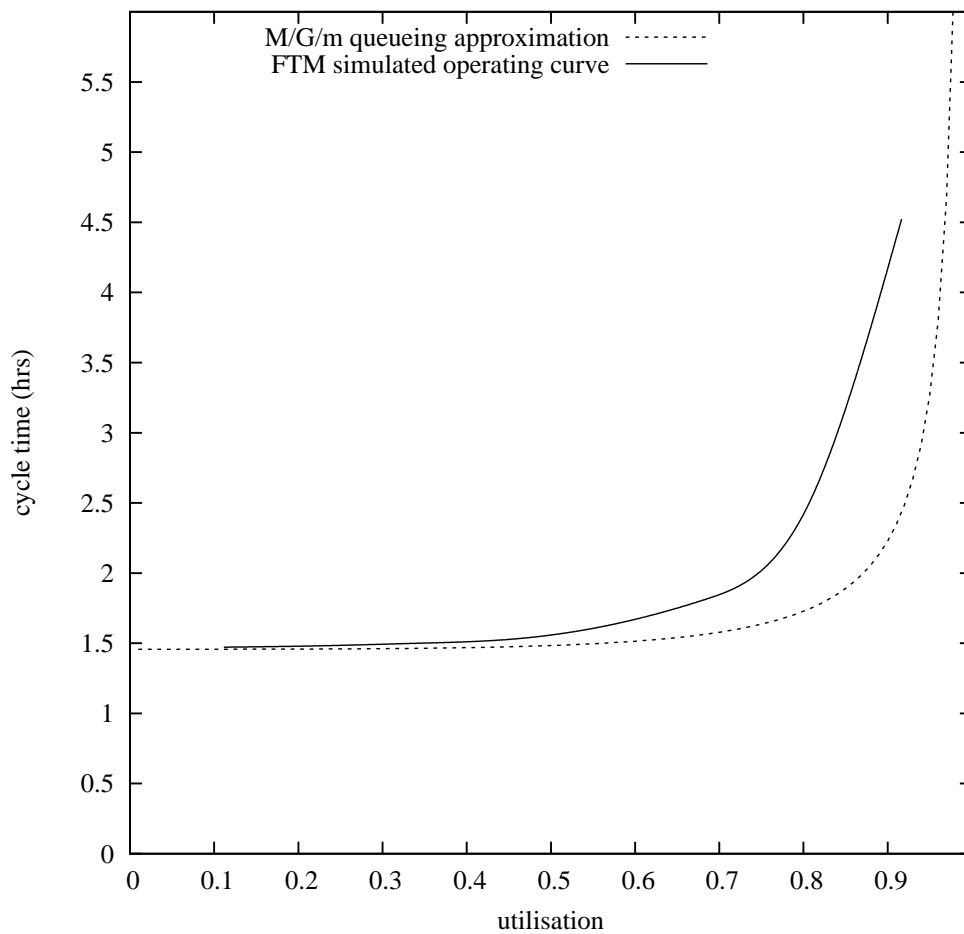


Figure 4.21: The operating curves generated by the simulation and the equivalent M/G/m queueing approximation for the system.

Table 4.7: Utilisation and cycle time predicted by the simulation model..

u	CT (hrs)
0.111501	1.472051
0.22566	1.481629
0.341053	1.499786
0.456686	1.530498
0.570868	1.630778
0.683033	1.811551
0.805738	2.486907
0.91656	4.522587
0.998243	71.99577 [†]

[†] The simulation did not achieve steady state.

Fig. 4.21 shows the calculated queueing theory approximation (see subsequent section) and the equivalent resulting operating curve given by the FTM application, based on values given in Table 4.7. Above a loading level of $u \approx 0.92$, the model became unstable and could not achieve steady state. Hence, the last design point was excluded from the analysis. The graph shows that the simulated curve increases faster towards the higher loading level and appears to take a steeper ascent at around $u \approx 0.75$, whereas, the queueing theory curve maintains its horizontal asymptote at higher loading levels.

4.6.1 Estimating the theoretical operating curve

Given that the lot inter-arrival times are assumed to be exponential and the process times are modelled using the Johnson family of distributions, the system operating curve can be estimated using an M/G/m queue. Note that Figs. 4.4-4.6 show that the exponential distribution is appropriate for this case study and Appendix F promotes the case for using the Johnson distribution as a good estimator for processing time distributions.

Assuming arrivals according to a weighted average of the arrival rate r_a for each operation i , as in Eq.(4.1), where π_i is the proportionate operation mix,

$$r_a = \frac{\sum_{i=1}^n \pi_i r_{a_i}}{n}, \quad \text{for } i = 1, 2, 3, \dots, n, \quad (4.1)$$

then the average utilisation u of the system can be calculated from the arrival rate r_a , the number of machines m , and the mean effective process time t_e according to,

$$u = \frac{r_a t_e}{m} \quad (4.2)$$

The mean effective process time t_e is calculated from the weighted average of all operations given by,

$$t_e = \frac{\sum_{i=1}^n \pi_i t_{e_i}}{n}, \quad \text{for } i = 1, 2, 3, \dots, n \quad (4.3)$$

The squared coefficient of variation for process time c_e^2 is calculated by using an operation weight factor. It is important to calculate c_e^2 for each operation using the operation's individual mean process time t_e and standard deviation σ_e and then averaging with a weight factor across the operations as in Eq.(4.4). If c_e^2 is calculated collectively for all of the operations, then the underlying stochastic mechanisms of the processing patterns could be lost.

$$c_e^2 = \frac{\sum_{i=1}^n \pi_i \left(\frac{\sigma_{e_i}^2}{t_{e_i}^2} \right)}{n}, \quad \text{for } i = 1, 2, 3, \dots, n \quad (4.4)$$

$$CT_{M/G/m} = t_e \left(1 + \left(\frac{1 + c_e^2}{2} \right) \frac{u \sqrt{2(m+1)-1}}{m(1-u)} \right) \quad (4.5)$$

Using these values and Eq.(4.5), an estimated operating curve can be found that is used as a guide to specifying the parameters of the simulation study (see Chapter 3). It is also useful for comparing the difference between the simulation model operating curve and the analytical curve derived from the queueing approximation.

4.7 Model Verification & Validation

Under normal circumstances the model can be verified using methods that mainly involve some sort of comparison with data from the real system. In the case of a flexible reusable generic model intended for a range of real systems, verification becomes a more difficult task. This section discusses how the model was verified for the case study only. Further verification tests would be required before it could be successfully deployed to other toolsets. Some of the verification techniques, summarised by Whitner and Balci (1989) and listed in Appendix G.1, were applied, and the results are briefly discussed in Table 4.8.

Table 4.8: Techniques used to verify the FTM application.

Type	Technique	Results and Comments
Informal	Walk-through	The structured walk-through was more heavily used as a validation tool (see Table 4.9) and used little to verify the program. This was due to a lack of personnel on-site that had both an intimate knowledge of the toolset being used in the case study and the programming languages used for the application (VB and ModL).
	Code inspection	The source code was reviewed and inspected during model build and after the final version. Much of the additional code included was to compensate for the irregularities in the source data.
Static	Syntax analysis	Modern software compilers ensured that the model syntax was complete and verified. The application code was compiled using Visual Basic for Applications (VBA) in Microsoft Excel. The ModL code was verified using ExtendSim's internal code compiler.
	Structural analysis	Structural analysis was performed according to the best practices of coding. Many comments were included with the code to ensure a clear and unambiguous intent of the code.
Dynamic	Top-down testing Bottom-up testing Black-box testing	Top-down and bottom-up testing was applied using a black-box style that checked each model hierarchy with dummy entities to ensure its portion was consistent and did not produce erroneous outputs.
	Stress testing	Lot arrival rates were gradually increased until the model became overloaded and unable to reach steady state. Also, MTBF and MTTR values were increased gradually causing very small tool availability. This resulted in very long queueing times in the model as expected.
	Debugging	Debugging was an ongoing process during the model build.
	Execution tracing	Data flow analysis and model tracing was performed for the simulation model by introducing only one lot into the system, and watching its attributes change during runtime. The lot experienced no queueing and minimal process time, proving that the model data flow was correct.

continued on next page

continued from last page

Execution monitoring	Trace animation capabilities in ExtendSim were used to validate the entity flow during runtime.
Regression testing	Regression testing (repeating the above procedures) was performed after each new model version.

The final model was validated using the validation techniques outlined in Table 4.9. Again, the model was found valid for the toolset under analysis, but further validation tests would have to be performed before declaring the model valid across a range of toolsets. Table 4.9 shows a summary of the tests that were performed on the model. An explanation of each test can be found in Appendix G.2.

Table 4.9: Techniques used to validate the FTM application.

Technique	Results and Comments
Animation	ExtendSim's animation allows various animation speeds that facilitated better understanding of the model. It also allowed entity icons to be changed depending on their status which was useful during the validation process.
Comparison to other models	Queueing model results were compared to ensure that the ExtendSim model was operating within normal ranges.
Degenerate tests	Degenerate tests were performed by gradually increasing the arrival rate of test lots into the model and forcing utilisation of the toolsets to capacity, making the model unstable and unable to attain steady state. This behaviour was expected and helped in part to validate the model and examine the boundaries of its operation.
Face validity	Face validity was one of the most frequently used techniques throughout the modelling process. Interviews and consultations with fab personnel was key to the construction of the conceptual model and ultimately the most effective method of validating the simulation model.
Historical data validation	Historical data records of cycle time data were compared to the cycle times reported by the simulation model. The results are shown in Fig. 4.22.
Internal validity	Model output data at the lower design points ($u \leq 0.8$) showed very little variation in the output performance metric across replications. This confirmed that results were consistent across runs and that the model was stable.

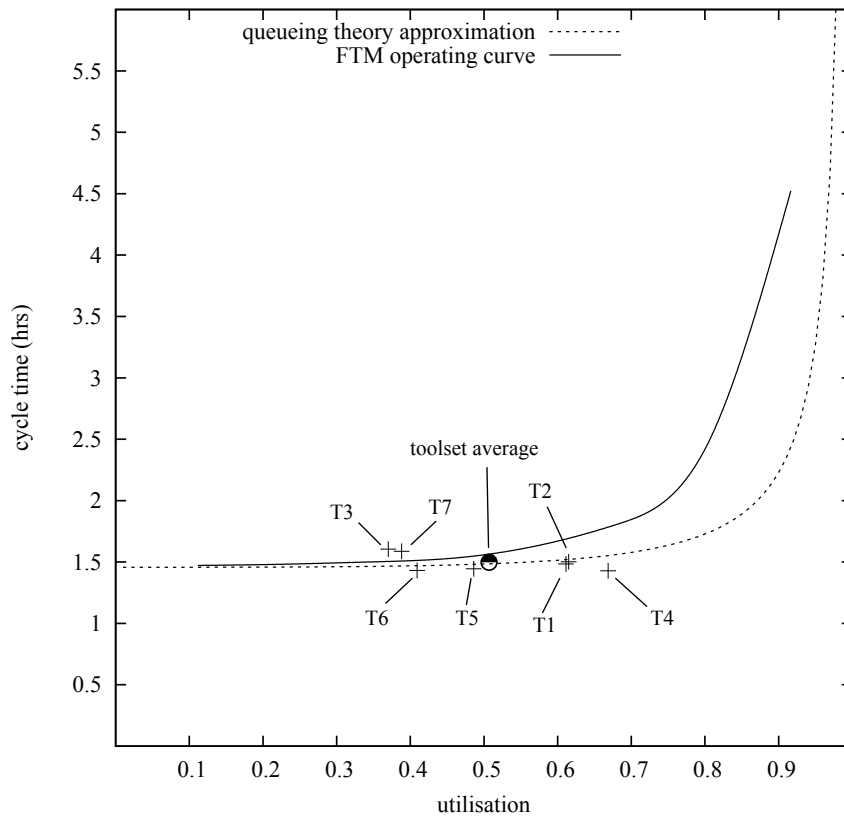


Figure 4.22: Tool operating points from historical records plotted alongside the queueing approximation and simulated operating curves.

4.8 IDEF0 Model Interpretation

Figures 4.24-4.30 show a set of IDEF0 diagrams that describe the model in depth. IDEF0 diagrams are usually used to capture the process flow in a system, although they are flexible enough to be adapted for other purposes. Here, they are used to describe the model from the viewpoint of the modeller, whereby the *resources* used by an IDEF process are the blocks that perform functions in ExtendSim, as in Fig. 4.23. The library that the block belongs to (denoted by *.lix* ending) is also given. There are three main libraries used; *item*, *value* and *custom*. The *custom* library holds all the blocks that were built to facilitate the FTM application in the absence of appropriate blocks provided by ExtendSim. Figure 4.31 also shows an IDEF1x diagram of the objects and attributes used to describe the entities in the model.

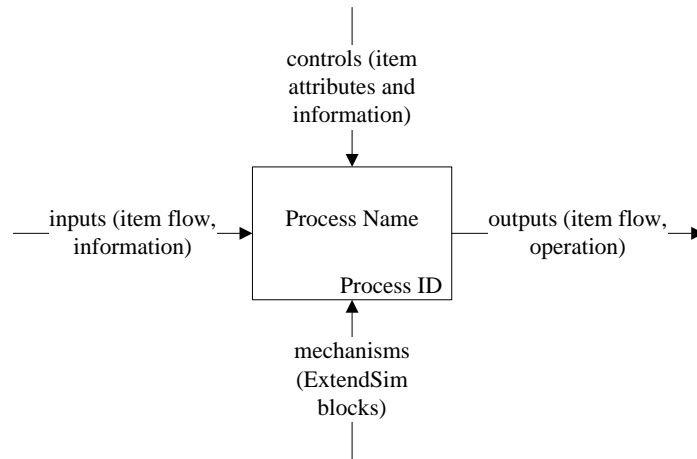


Figure 4.23: IDEF0 standard adapted to the viewpoint of the modeller.

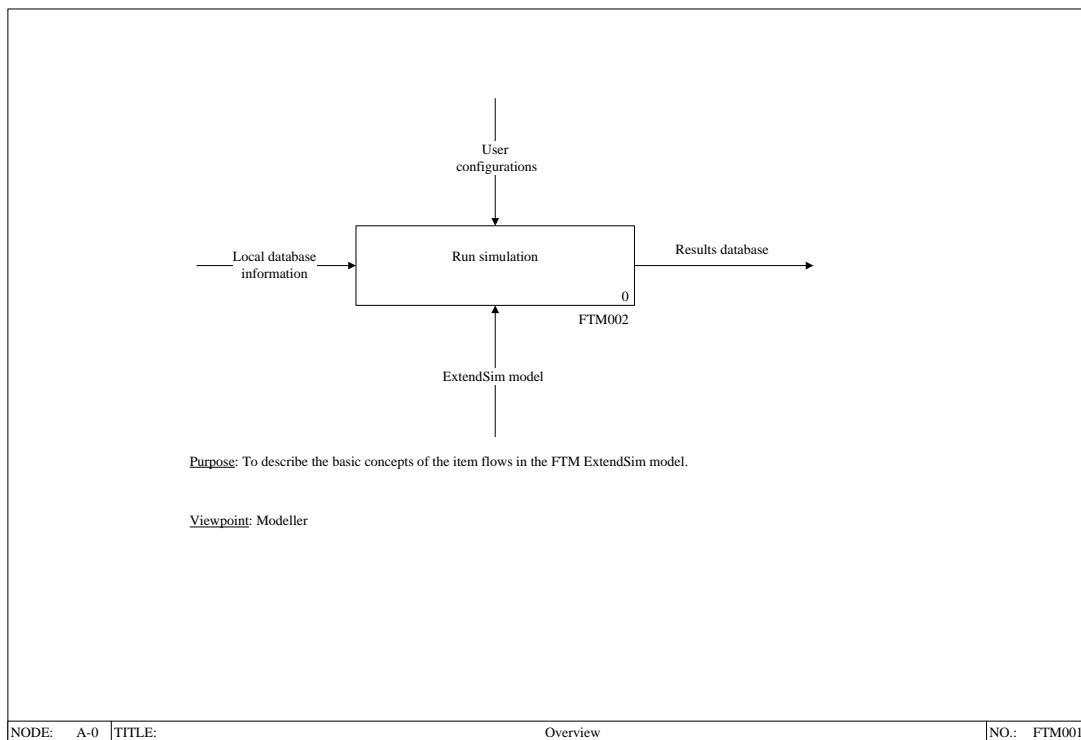


Figure 4.24: Overview IDEF0 diagram (A-0) for FTM ExtendSim model.

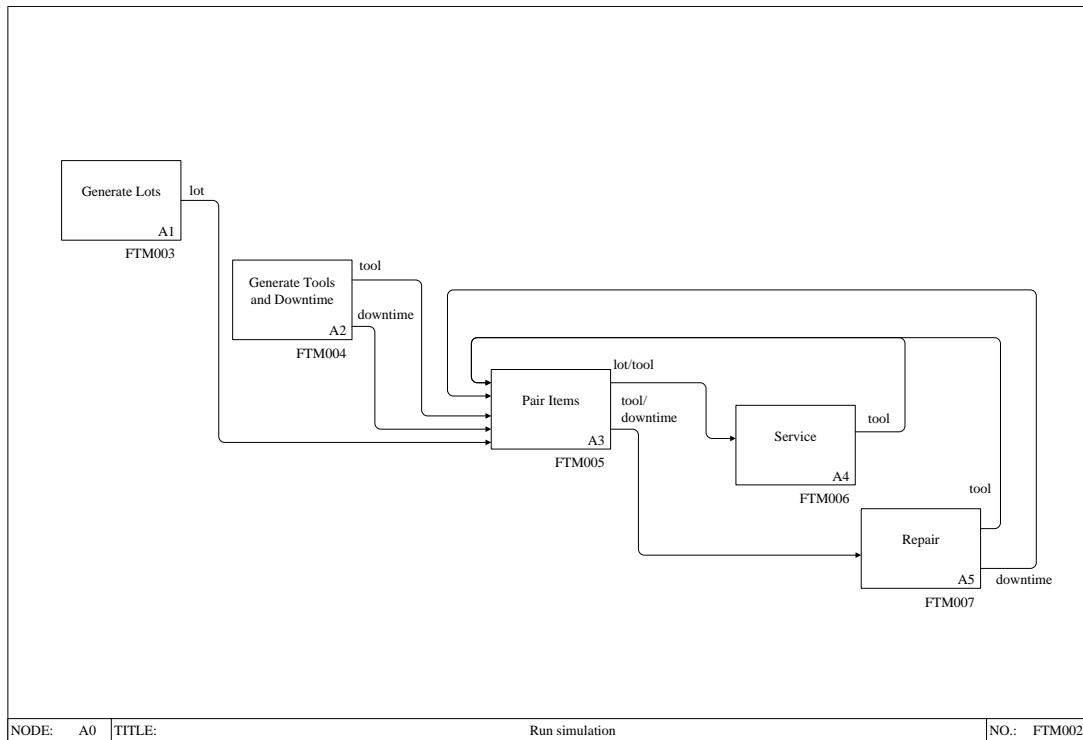


Figure 4.25: 'Run Simulation' (A0) IDEF0 diagram for FTM ExtendSim model.

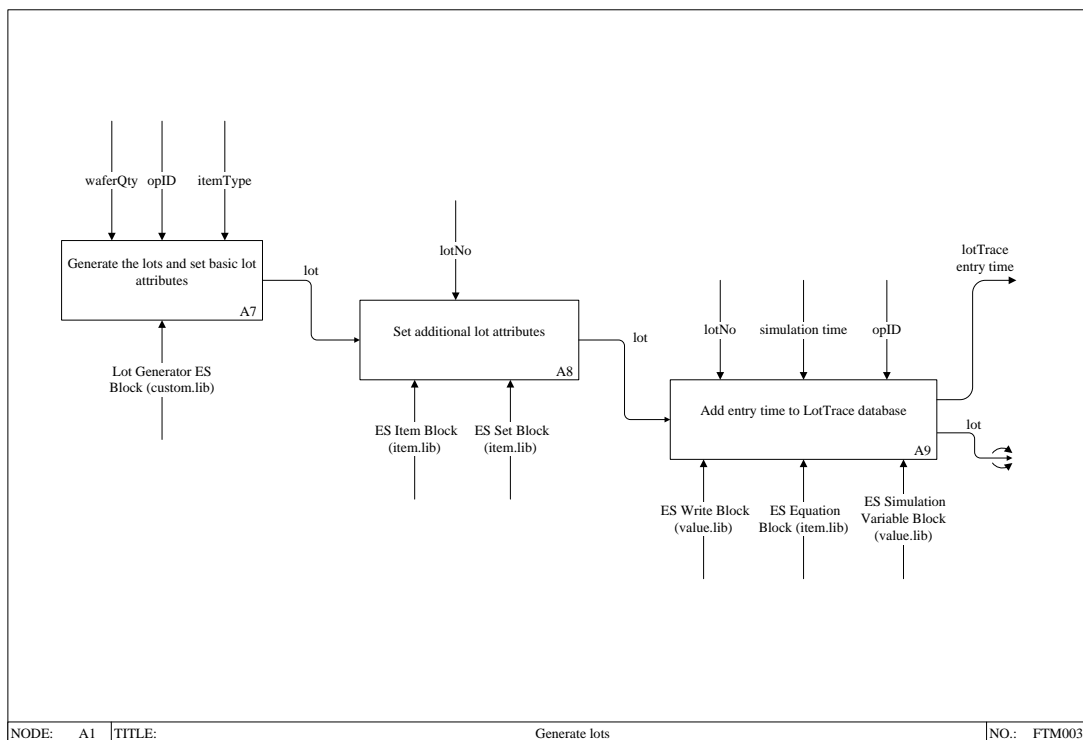


Figure 4.26: 'Generate Lots' (A1) IDEF0 diagram for FTM ExtendSim model.

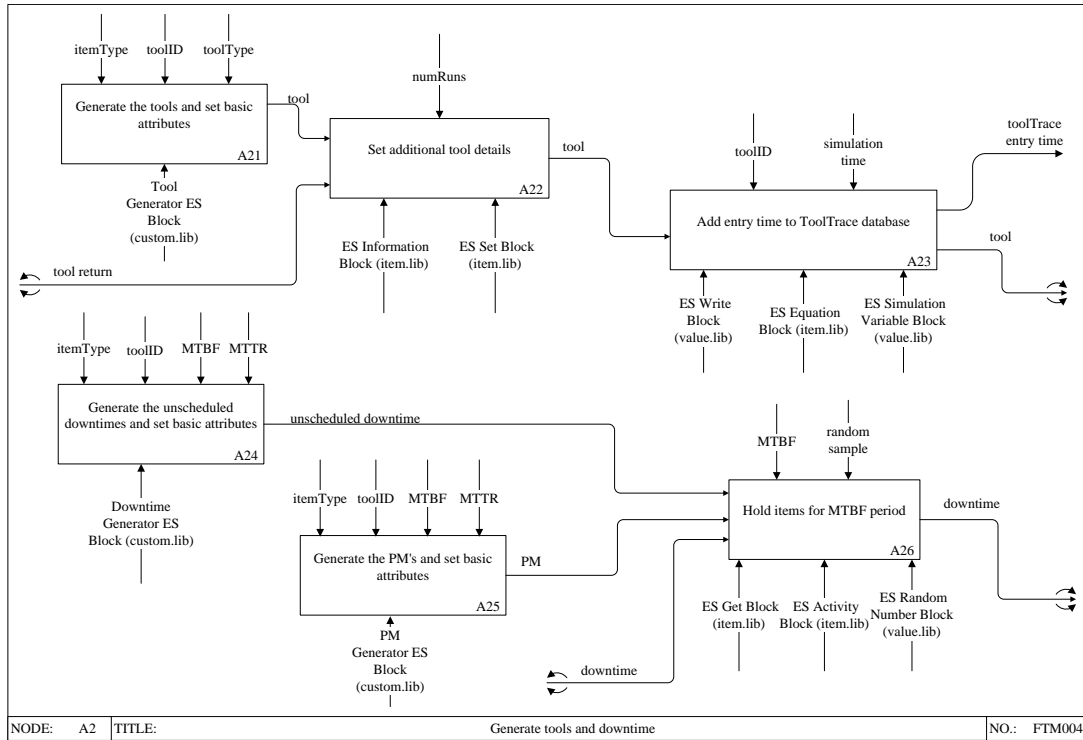


Figure 4.27: 'Generate Tools and Downtime' (A2) IDEF0 for FTM ExtendSim model.

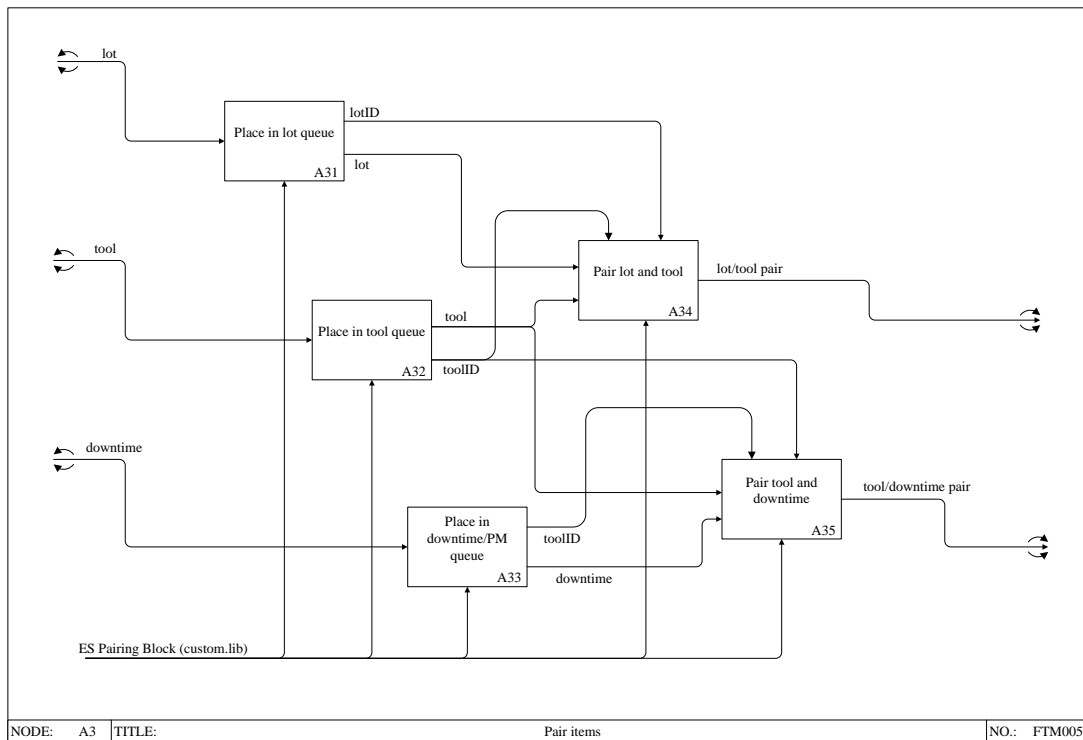


Figure 4.28: 'Pair Items' (A3) IDEF0 diagram for FTM ExtendSim model.

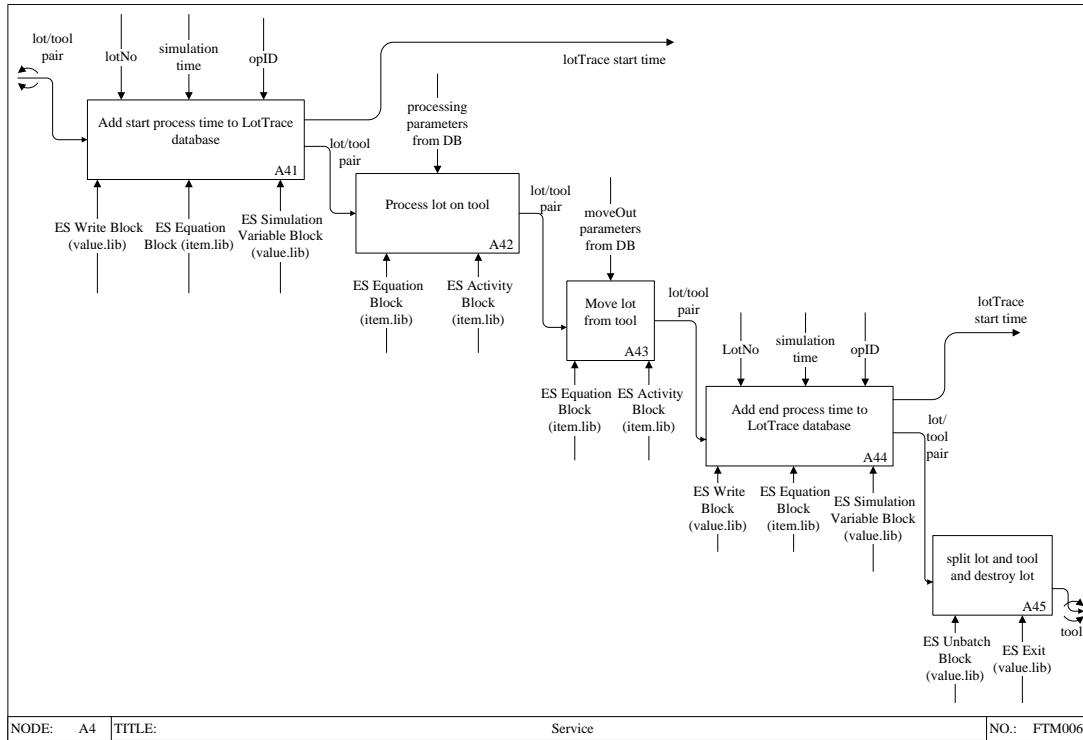


Figure 4.29: 'Service' (A4) IDEF0 diagram for FTM ExtendSim model.

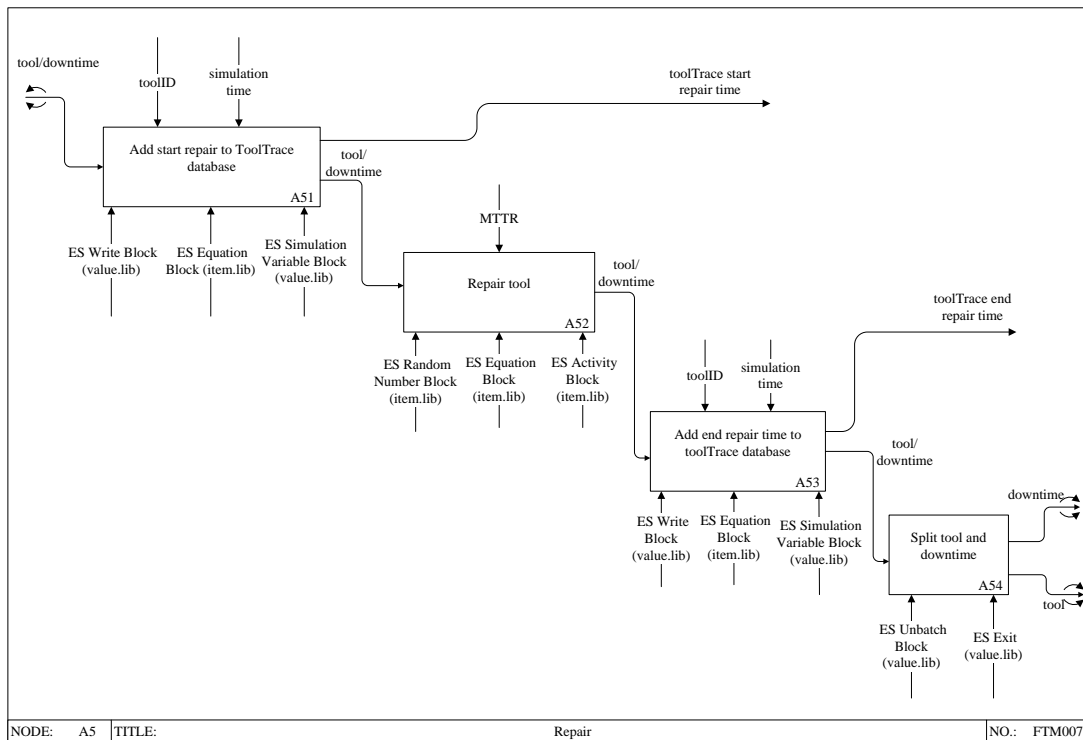


Figure 4.30: 'Repair' (A5) IDEF0 diagram for FTM ExtendSim model.

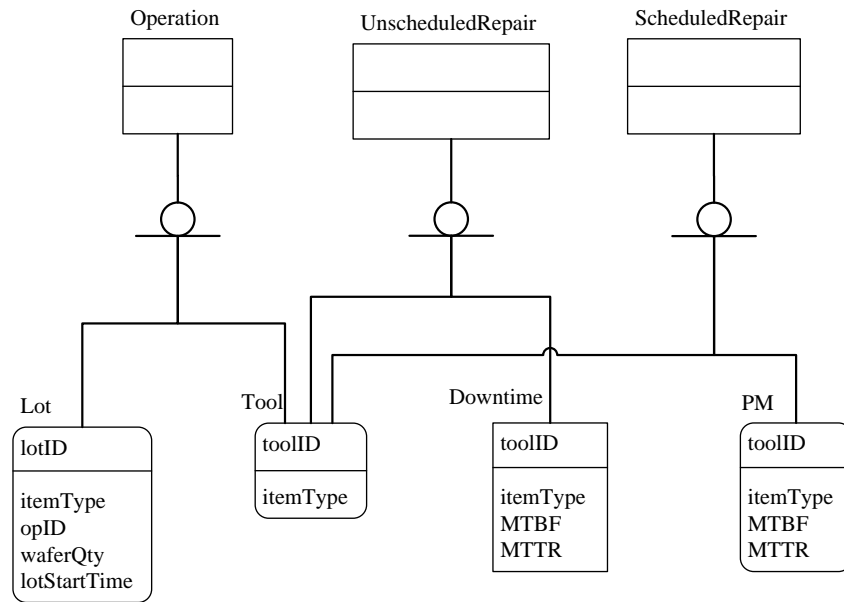


Figure 4.31: IDEF1x diagram showing the objects and attributes used to describe the entities in the FTM ExtendSim model.

4.9 Summary

This case study showed that the FTM application can be successfully used to generate operating curves for toolsets or functional areas in a semiconductor fab. One of the largest difficulties during development of the application was processing the raw data retrieved from factory databases. Despite an abundance of data being available and the relative ease of extraction, there were many issues over interpretation of the data. Understanding what actually happened in the fab from the historical data timestamps was a difficult and a very time consuming process. Any tool errors or breakdowns often resulted in confusing entries in the historical records that were difficult to legislate for in the program. Nevertheless, the concept of the application was proved, and it provides a basis and framework that would allow such an application to be further tested or ultimately rolled out factory-wide.

In general, the theoretical operating curves based on queueing approximations seemed to show that the tools are efficient and exhibit a very low variation in process times for each operation. This is expected given that the tools are highly automated and the processes they perform are strictly controlled. Some of the model validation methods

exposed other local decision-making factors that were not anticipated and could not be accounted for in the model. For example, it could not account for artificial queueing at the toolset that was implemented by operators if a downstream toolset went offline. Instead, the model assumed that outside factors were independent. This is not always the case, and it was found that operators attempted to share the burden of queueing by withholding lots in some situations.

Similarly, re-entrancy could not be effectively or realistically modelled by a single toolset simulation, particularly if the re-entrant loop passes back through the same toolset soon afterwards. In the real system, the tool availability and departure variability of a highly re-entrant toolset would have a large impact on the arrival pattern to the toolset and the model could not account for this behaviour.

Therefore, it was concluded that analysis of a toolset in isolation might be insufficient, and it was decided to produce a flexible reusable model for generating operating curves on a factory-wide scale, where re-entrancy and other such factors could be captured more effectively.

Semiconductor Fab Model A

There are many issues involved in implementing a simulation study of a semiconductor fabrication plant. Primarily, the complexity and the scale of the real system means that collecting data, building the model and interpreting the results can have a very long lead time. This chapter describes a discrete event simulation model that attempts to overcome these issues and reduce the project lead time by being fully flexible, automated and reusable. The aim of this model is to allow engineers to experiment with a variety of configurations and production strategies in the fab. The structure of the model is based on the *Semiconductor Wafer Manufacturing Format Specification*, created by Feigin et al. (1994), that outlines an information model for creating sample datasets from semiconductor factories. The discrete event simulation (DES) model was created using ExtendSim modelling software and is encapsulated in a Visual Basic (VB) application. The application constructs an experimental framework, generates a simulation model and displays the factory operating curve. The result is a full simulation framework that

automates many of the time consuming tasks and allows the user to compare alternate systems and/or operational policies, rapidly and confidently. Finally, this chapter examines a dataset provided by the specification and shows how the operating curves from the flexible modelling application can be used to identify an potential weaknesses in the fab and offer some suggestions on how to minimise cycle time and maximise the system efficiency.

5.1 Semiconductor Wafer Manufacturing Data Format Specification

The *Semiconductor Wafer Manufacturing Data Format Specification* was formed to address the lack of factory level representative data available for academics and industrial engineers to experiment with product flows and compare fab specifications. Currently there exists eight sample datasets, some of which have been constructed by the authors of the format and others that have been donated by anonymous fabs. The datasets are available to download from http://wwwalt.sim.uni-hannover.de/~svs/wise0910/pds/masmlab/factory_datasets/. The format consists of six files per dataset. The purpose of each file is listed in Table 5.1 and further information about what is contained in the files can be found in Tables C.2-C.6 in Appendix C.

Table 5.1: Data files used for wafer data format specification.

<i>File</i>	<i>Suffix ID</i>	<i>Description</i>	<i>Reference</i>
Process Route	pr	Process route information for all processes	Table C.2
Rework Sequences	rw	Information on rework sequences	Table C.3
Tool Set	ts	Information on tools	Table C.4
Operator Set	os	Information on operators	Table C.5
Volume Release	vr	Release rate information	Table C.6
Comment File	cf	General comments and sample run results	n/a

The volume release file divides product groups into specific recipes known as process flows, and describes how the product is released into the factory. The process route file

details the operations list (or steps) that each process flow follows. Each step contains processing information for a particular operation including batching and setup requirements, type of operator, toolset and processing pattern. It also contains post-processing information such as yield and rework probabilities and transport mechanisms.

The rework sequence file is very similar to the process route file and contains all the processing and routing information for lots that must undergo a rework path. Once this rework path is complete, lots rejoin their previous process route.

The operator set file contains information about the quantity of operators and their break requirements. Similarly, the toolset file contains tool information such as tool quantity per toolset, wafer-based and time-based downtime patterns (maximum of five) and the percentage time required by operators for each processing phase on the tool. The comments file contains summary results from sample simulations and any other general information about the factory.

5.2 Project Objectives

In the initial scoping phases of the project, some objectives/requirements were laid out such that the model would be automated, user friendly, flexible and reusable. These requirements are listed as follows,

1. The input analysis (experimental setup) must be fully automated and statistically sound,
2. The output analysis should be automated, graphical and compare operating curves,
3. Users should not require any 'expensive' software,
4. The model should be completely self-contained and not require any alteration or interaction from the user,
5. The model should display animation (if required) for model validation and verification purposes,
6. The model should be documented using a consistent systems modelling method (e.g. SysML or IDEF diagrams).

The model is designed to replicate real phenomena associated with semiconductor fabs such as tool setups, lot scrapping, downtime, rework, re-entrancy, tool diversity and varying product routing (as discussed in Section 2.1.3). This model aims to show how the full specification can be implemented using an efficient reusable simulation modelling framework that can address any factory configuration posed by the information model.

5.3 Modelling Strategy

Modelling using the traditional job-driven method (described in Section 4.4) creates a complex and rigid model with multiple routes and connections. Model size in graphical simulation packages is generally dictated by the number of blocks in the model. These blocks hold many lines of (often superfluous) code. Therefore, the lesser the number of blocks, the less code to be executed and ultimately, the faster and more efficient the model will be. For example, dataset 6 has over 100 toolgroups (see Table C.7 in Appendix C). If the average number of tools per group is even three, then the number of tools for the dataset is about 300. Hence, the model would require at least 300 blocks to model the tools. In fact, this figure is more likely to be about three or four times this when one considers the periphery blocks usually required to control the tool blocks in ExtendSim.

Table 5.2: Comparison of the traditional use of basic simulation objects and an entity-centric approach to model a semiconductor fab.

Article/Event	Traditional approach	Entity-centric approach
Lots	Modelled as <i>entities</i> , created using <i>generators</i> and deleted by <i>destroyers</i> .	Modelled as <i>entities</i> , created using <i>generators</i> and deleted by <i>destroyers</i> .
Tools	Modelled as <i>locations</i> .	Modelled as <i>entities</i> , created using <i>generators</i> and deleted by <i>destroyers</i> .
Operators	Modelled as <i>resources</i> .	Modelled as <i>entities</i> , created using <i>generators</i> and deleted by <i>destroyers</i> .
Downtime Events	Modelled as <i>resources</i> .	Modelled as <i>entities</i> , created using <i>generators</i> and deleted by <i>destroyers</i> .
Buffers and Stockers	Modelled as <i>queues</i> .	Modelled using single <i>queues</i> .
Material Handling Systems	Modelled as <i>resources</i> .	Modelled as <i>entities</i> , created using <i>generators</i> and deleted by <i>destroyers</i> .

Therefore, it is not possible to create a flexible and efficient model with this strategy. An alternative technique, *entity-centric modelling*, used in the Flexible Toolset Modelling (FTM) application discussed in Chapter 4 involves using less blocks and modelling as many objects and events using entities, as shown in Table 5.2. The entities do not hold any program code, the only information required is attributes that describe the entity. The model keeps an internal storage list of the location of each of the entities circulating the model. By removing the notion that entities are widgets that circulate the real system, and moving towards a conceptual model where entities such as lots, batches, tools, downtime events and repair events circulate the system and mate with other entities to perform a task, the model becomes more flexible.

Table 5.3: Entity-centric approach to modelling semiconductor fabs.

Article/Event	Simulation modelling structure
Lots and Batches	Modelled as <i>entities</i> , created using <i>generators</i> and deleted by <i>destroyers</i> .
Tools	Modelled as <i>entities</i> , created using <i>generators</i> .
Processing	Lot and tool entities are paired and delayed in an infinite capacity <i>location</i> .
Operators	Modelled as <i>entities</i> , created using <i>generators</i> , delayed at infinite capacity locations to represent operator jobs or breaks.
Downtime	Downtime items modelled as <i>entities</i> , created using <i>generators</i> .
Buffers and Stockers	Modelled as common infinite <i>queues</i> .
Material Handling Systems	Modelled as <i>entities</i> .

By pairing and splitting these entities, it is possible to reduce the size of the model down to only a few blocks. These blocks are circulated by entities that represent a host of events, articles and structures of the real system. Table 5.3 shows how the entities are joined, delayed and split to represent activities in the fab.

Fig. 5.1 shows an IDEF1x interpretation of the objects and entities in the models. There are five atomic (indivisible) types of entities that circulate the model; *lot*, *tool*, *downtime*, *operator* and *break*. From these, a number of other *superclass* entities can be created. A *batch* superclass consists of a number of instances of the *lot* class. A *process* superclass consists of a *tool* instance paired with either a single *lot* or with a *batch* instance. A *repair* superclass is made up of a single instance of a *tool* and a single

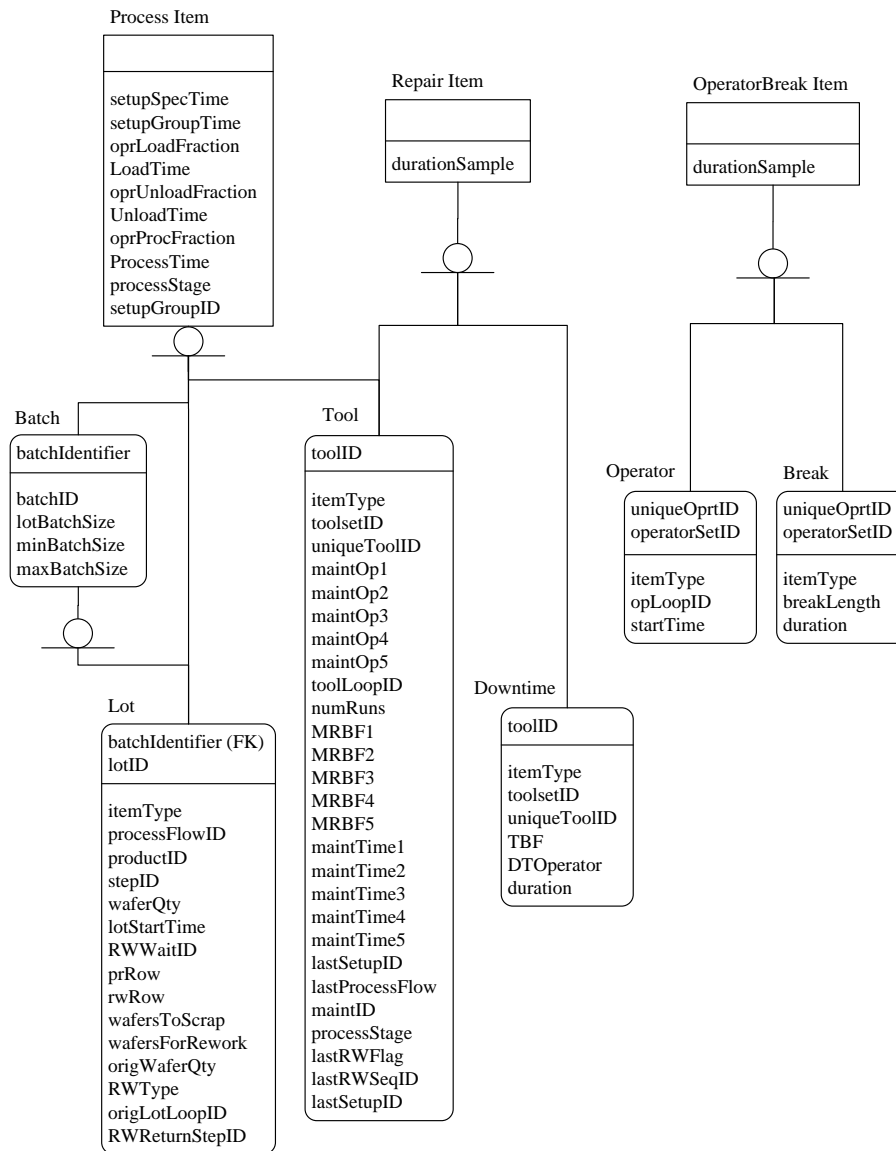


Figure 5.1: Modelled entities and their attributes described using IDEF1x.

instance of a *downtime* class, and an *operatorbreak* instance consists of single instances of an *operator* and *break* class. The instances formed during model runtime, represent the state of the *child* classes. For example, an instance of a process superclass means that its child *lot* instance is undergoing an operation on the other child member, an instance of the *tool* class.

5.4 Model Input and GUI

The model is controlled from a VB application that invites the user to input their choice of dataset (Fig. 5.2). Other selections that can be input include the release pattern, number of warm up increments and the simulation run length (Fig. 5.3).

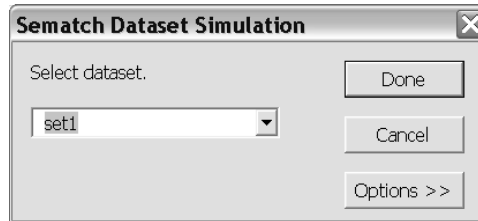


Figure 5.2: Dialog option available for user to select a dataset.

To increase the anonymity of the Sematech datasets the volume release file has no release pattern implied or otherwise. To compensate, five options are available for the user when selecting an appropriate release rate; twice daily, daily, weekly, fortnightly and exponentially distributed. The first four are based on a constant output after every defined period of time, the 'exponentially distributed' option computes the average wafer starts per hour based on the volume release per week and uses this average to construct an exponentially distributed lot inter-arrival process for the model.

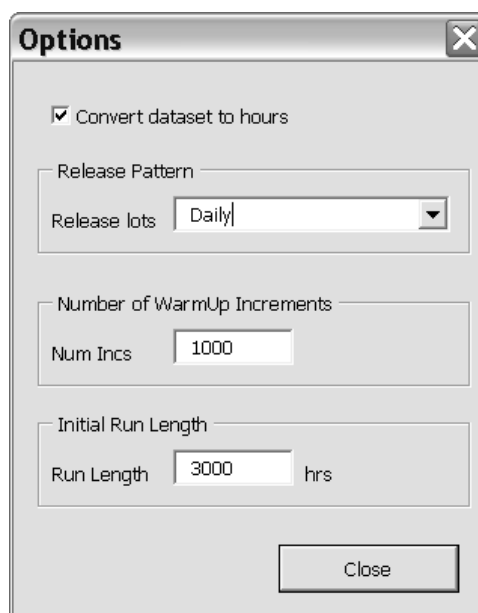


Figure 5.3: Additional release pattern options for selection.

Other non-default options include the warm-up period increment which aids the smoothing of the output from the simulation performance measure (cycle time). If the simulation is overloaded, that is, there is a surplus of input over capacity, then a plot of the cycle time output is usually a ‘smooth’ straight line heading to infinity with very little ‘noise’. Although one could never say for certain, typically, such an output is indicative of a model that would never attain steady state, irrespective of how long the model is run for. Therefore, it is necessary to allow the user to visually check that this is not the case, by evaluating the cycle time output from the model. This issue is discussed in greater detail in Section 5.6.7.

5.5 Communicating with ExtendSim from VB

The ExtendSim simulations are controlled from the VB program and a number of functions and subroutines were created as a wrapper for ExtendSim which allows the program to populate the database, run the simulations as a background process and extract the simulation outputs. The wrapper functions are listed in Table 5.4 and the code is included in Appendix B.3.

5.6 Model Description

The program begins by pulling the text based datasets from a local source, and formatting them for ExtendSim’s database. The subroutines and programs required for this process can be found in Appendix D.1. The following section describes how the system model is generated.

5.6.1 Lots and batching

Lots are created according to the volume release data file and the release pattern selected by the user. Lots are assigned a Product ID and a Step ID which makes its next operation

Table 5.4: VB wrapper functions for ExtendSim.

Function	Description
RetrieveModel	Opens the ExtendSim model file.
GetExtendAppPath	Finds the local ExtendSim installation.
PassAllDataToExtendSim	Sends all the data and distribution information from VB to ExtendSim.
RunExtendSimModel	Runs the model.
SaveAndCloseExtendSimModel	Saves the model and cleanly exits ExtendSim.
PassArrayToExtendSim	Creates an ExtendSim array and populates it with the contents of a VB array.
PassRunInfo	Informs ExtendSim of some essential run parameters such as start time, end time and the number of replication to perform.
ExtendDBTableWrite	Writes an ExtendSim database to a text file in the root folder of the ExtendSim application.
ReceiveDBfromExtendSim	Creates an array in VB and populates it with an ExtendSim array.
ReceiveArrayfromExtendSim	Converts a VB array to string and passes it to a newly created ExtendSim array.
SimAnimation	Turns the simulation animation on or off.

a unique step. Once lots have completed all their operation steps listed in the process route file they are exited from the system. Note that the model is not pre-populated or initialised with any pre-existing lots, and that the system starts from empty.

During model execution, the lot entities reference the process route file in the database to find the resources needed for their current operation. The resources for the lot always include the toolset necessary to complete the operation, but it may also include transport operators and/or processing operators. If the lot is to be processed on a toolset that supports batching, i.e., a batch tool, multi-sequence tool, conveyor tool, cluster tool or linked-track tool (see Appendix C.3), then the lot will wait for an appropriate batch size to be formed. Two batching policies were examined; minimum and maximum batch sizing. Selection of batch sizing has a very large impact on results, and the operating curve. A lightly loaded fab favours a minimum batch size policy, whereas a heavily loaded fab favours the maximum or full batch size policy. This is not surprising given that many of the datasets contain heavily re-entrant process steps through toolsets that have long process times and large maximum batch sizes.

Equation (5.1) can be used to calculate the degree of re-entrancy (DoR) (Ignizio,

2009) and Table 5.5 shows the DoR for each of the datasets. The DoR, is used only as an indication of the complexity of the fab, however, most of the datasets were shown to have a very high DoR.

$$\text{DoR} = \frac{\text{no. of operations}}{\text{no. of toolsets}} \quad (5.1)$$

Table 5.5: Degree of re-entrancy for *Semiconductor Wafer Manufacturing Data Format Specification* sample datasets.

dataset	no. of operations	no. of toolsets	DoR
minifab	18	5	3.6
1	455	83	5.48
2	1606	97	16.56
3	4138	73	56.68
4	111	35	3.17
5	4176	85	49.11
6	2541	104	24.43
7	172	24	7.17

The maximum and minimum batch sizes are dictated by the maximum and minimum wafers per batch in the datasets. The minimum and maximum lots per batch can then be calculated given the starting lot size, as in Eqs (5.2) and (5.3). The lots wait until a batch is complete before requesting a tool from the toolset to perform the required step or operation. Lots can be batched together if they have the same Batch ID. If the batch ID is omitted from the dataset, then only lots from the same process step on the same process flow can be batched together.

$$\text{BS}_{\min} = \left\lceil \frac{\text{minimum wafer per batch}}{\text{wafers per lot}} \right\rceil \quad (5.2)$$

$$\text{BS}_{\max} = \left\lfloor \frac{\text{maximum wafer per batch}}{\text{wafers per lot}} \right\rfloor \quad (5.3)$$

5.6.2 Lot processing

Tool ‘occupancy’ commences once a lot or batch has captured a tool. The first step is to check if the tool requires a setup for the impending operation. The setup requirement of the tools is based on two mechanisms;

- If the last lot processed on the tool is from the same process flow as the current lot, then a setup based on the `Time per Spec Setup` time is required,
- If the last lot processed on the tool was from a different setup group ID, then a setup based on the `Time Per Group Setup` is required.

Next, the lots are loaded onto the tool. Loading may involve an operator for a certain time fraction of the loading process. Once the lot/tool pairing has captured an appropriate operator, the operator is then occupied for the `Operator Loading Fraction` of that particular operation. Once the loading fraction of time has been reached the operator is released.

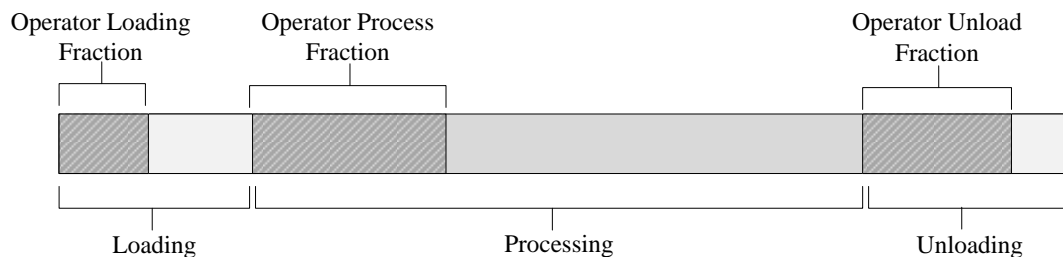


Figure 5.4: Portion of time the operator is occupied during the process step.

The next step, processing, operates in a similar way as loading in that an operator is required for a fraction of the process time. Unloading the lot from the tool follows the same pattern again. In this way, an operator may be called up to three times for one process step as shown in Fig. 5.4. It is also assumed, (because it is otherwise undocumented), that the fraction of time that the operator is occupied for either loading, processing or unloading, is at the beginning of the task. An argument could be made that the unloading fraction of the operators time is towards the end of the unloading process, however, this would be far more difficult to implement in the model. Furthermore, unloading time was generally very small in comparison to the processing step, and

the operator unloading fraction was even smaller. Based on this, and due to a lack of clarification in the dataset documentation, it was decided that it was sufficient to allocate the proportion of operator unloading time to the beginning of the activity.

Equation (5.4) is used to calculate mean process time per lot pt , which is then used to calculate the time until tool becomes free tf in Eq.(5.5), and total lot cycle time through an operation ct in Eq.(5.6):

$$\begin{aligned} pt = & \text{Time per Batch} * \text{No. of batches required for the lot} \\ & + \text{Time per Lot} \\ & + \text{Time per Wafer in Process} * \text{No. of wafers in the lot} \\ & + \text{Product Setup (if appropriate)} \\ & + \text{Group Setup (if appropriate)} \end{aligned} \tag{5.4}$$

$$tf = \text{Load Time} + pt + \text{Unload Time} \tag{5.5}$$

$$ct = \text{Load Time} + pt + \text{Wafer Travel Time} + \text{Unload Time} \tag{5.6}$$

The total time the lot is occupied (ct) differs from the time the tool is occupied (tf) by the addition of the `Wafer Travel Time` which is effectively the cascade time on the tool. A consequence for the model strategy used here, where lot and tool items are paired together, is that the lot and tool items must be separated for the `Wafer Travel Time` period. The only way to account for this difference between tool occupancy and lot occupancy is to unpair the items and allow the tool to become available for other lots, while holding the lot for the `Wafer Travel Time` period. This is one of the disadvantages of using an entity-centric modelling approach. However, given that only a small proportion of the tools have a `Wafer Travel Time`, it was seen to have a negligible impact on the factory operating curve.

After the operation, the lot or batch is divided and transported to its next step, a task which may require a transport operator. The following steps summarise the interaction between lots, batches, tools and operators;

1. Lot begins a new operation step,
2. If it is a batch operation then lot waits to form a maximum or minimum batch size depending on the user selection,
3. Lot/batch queues for a tool from the required toolset,
4. If a tool is available then the lot/batch and tool are paired, otherwise the lot/batch queue for the next available tool,
5. Once a tool is captured, the lot/batch/tool request a loading operator, if one is available then the lot/tool/batch grouping are paired with the operator, otherwise the it waits for the next available operator from the operator set to become available,
6. The lot/batch/tool/operator grouping is held for the operator loading fraction,
7. The operator is then released and the lot/batch/tool grouping is held for the remaining loading fraction,
8. Steps 5-7 are repeated for the processing step and the operator processing fraction,
9. Steps 5-7 are repeated for the unloading step and the operator unloading fraction,
10. The tool is then released from the lot/batch grouping,
11. The lot/batch is held for the wafer travel time component to simulate the cascading time on the tool,
12. The batch (if one was formed) is then split into its original lots,
13. Each lot then carries out steps 1-12 for its subsequent operations until all of its required operations have been fulfilled.

5.6.3 Tool downtime

There are up to five different downtime or maintenance cycles allowed for each toolset according to the data specification (although adding more if needed would be a relatively trivial matter). Each tool within the toolset is assumed to have identical failure/maintenance pattern. Each individual downtime contains three pieces of information;

- Indicator whether the cycle is time-based or wafer-based,
- Mean time before failure (MTBF) if the cycle is time-based or the mean wafers before failure (MWBF) if the cycle is wafer-based,
- Mean time to repair (MTTR).

There are no specific references to any distribution patterns from the seven sample datasets, and the specification guide states that “No distributional information is included in the data sets beyond the first moment (mean) information” (Feigin et al., 1994). For reasons that are outlined in Section 2.4.1, it was decided that the downtime and repair patterns should be based on the exponential distribution. An additional benefit in using the exponential distribution is that the mean is the only required information, and it did not require speculation on additional parameters such as the standard deviation. This was a large factor in deciding distribution patterns for the MTBF, MWBF and MTTR.

Time-based failure

For each time-based failure, for each tool, a downtime item is created in the ExtendSim model. These downtime items are generated by the custom Tool Generator block and enter the model at time zero. They then wait in a delaying mechanism which holds them until a sample from their MTBF exponential distribution expires.

At this point, the downtime item attempts to locate and pair with its equivalent tool item identified by the matching unique tool ID from the same toolset ID. Once the items have been paired they are sent down another delay path and held for the period of repair specified by a sample from the MTTR exponential distribution. The repair event usually requires an operator or technician from a specified operator set, and will wait until one becomes available if not already available. The operator/technician is required for the full duration of the maintenance/repair event. Once the maintenance/repair event has completed, the operator resource is released, and the tool and downtime item are unpaired. After this, the tool returns to its normal cycle in the model and the downtime item is reset and returned to the start of the model where it will wait until the next

MTBF sample time expires. This process is repeated during runtime and proved an effective method to model multiple time based failure/maintenance cycles for a tool.

One disadvantage of this method occurs when a tool is processing a lot/batch and its equivalent downtime item's MTBF expires. In this situation, the tool would fail in a real world system, whereas in the model presented here, the tool has only really 'failed' when the downtime item has captured and paired with its mating tool item. Consequently, the downtime item must wait until the tool becomes available after processing a lot/batch. In the real system, this would be equivalent to pushing a 'failure' out until the current processing cycle has finished. This means that the actual resultant MTBF (as recorded by the simulation model) may be slightly greater than that of the intended or listed MTBF in the dataset because the downtime item must wait until its tool has finished processing. However, this was seen as having minimal impact on the model results, for the following reasons;

- (i) Given that the average MTBF times are, in general, far greater than the average process time, (often in a ratio of 10:1), the downtime item usually does not have to wait very long in comparison to how long it has been suspended in MTBF stasis,
- (ii) The previous point is even more emphasised if the tools have a low utilisation, meaning, the probability that the downtime item will find the tool engaged in processing is less given that the tool is mostly idle. Many of the tools in the datasets (usually about 70%) were found to be lowly utilised according to the volume release rate,
- (iii) Depending on how one views the failure/maintenance cycle, it may not be uncharacteristic of the real system that the repair event must wait until after processing has been completed. For example, although the datasets make no mention of exactly what type of offline event the cycle is modelling, in some circumstances, it is likely that these could be preventative maintenance (PM) or maintenance events, and it is very uncommon for processing to be interrupted by an impending PM procedure or task.
- (iv) Often unscheduled downs are the result of a statistical control issue, i.e., not a failure, but a request to go down owing to an out of control signal or trend. In such

cases, the machine/tool would complete processing of the lot and then enter the down state.

Wafer-based failure

Wafer-based failure refers to a failure pattern based on the number of lots that have been processed by the tool (also known as wear-based failure). This was modelled by attaching a `wafers-processed` counting attribute to each tool that is updated after every process step. When the MWBF quota is reached (which is evaluated post-processing of the lot), the tool is sent down a repair path and captures an operator resource (if one is required). Once repair is complete the operator is released, the tool item returns to its normal cycle and its `wafer-processed` counter is reset. Wafer-based offline events were easier to model than their time-based equivalents and no additional downtime items were needed. The wafer count check being carried out post-processing is justified as it is highly unlikely that tools would ever be taken offline mid-process to fulfil a scheduled wear-based PM.

5.6.4 Operators and breaks

Operators are generated by the custom Operator Generator block at simulation time zero and are added to an operator delay or queue where they are held in stasis until required. Periodically, either a tool/lot or tool/downtime pair will enter this queue and search for an appropriate operator to *pair* with for the fulfilment of a task (be it, processing or repair/maintenance). A possible solution, was to actually bind the operator item with its host for the duration of the task. However, this was infeasible given that in some circumstances the operator is only required for a portion of the task, as discussed in Section 5.6.2. A more complicated solution was implemented, whereby the operator would be released from stasis and sent down a delay path to represent the fraction of time it was occupied. This meant that the task duration needed to be pre-sampled before the

host pair engaged with the operator. The sample could then be used by the operator to work out the amount of time it was partially (or fully) occupied by the task. The steps are given as follows;

1. If the host item is a tool/downtime pairing then a MTTR sample is taken, if it is a tool/lot requiring either loading, processing or unloading, then a sample of the appropriate duration is taken,
2. The tool/lot or tool/downtime pairing that requires an operator joins the operator loop and requests an operator from the required operator set,
3. If an operator is not available the host pairing waits until one becomes available,
4. If an operator is available then both the host pair and the operator are held separately for the sampled duration or a fraction of it in the case of a loading, unloading or processing task,
5. Once the duration has expired the host group returns to its main loop and the operator returns to its stasis queue where it can be extracted by another host.

Break items are also generated by the Operator Generator block and are placed in a delay for a duration according to the `time between breaks` attribute. It was assumed that breaks are usually at a constant time and for a constant duration therefore, a deterministic `time between breaks` and `break duration` was used. Once the `time between breaks` has expired, the break item will capture the operator from stasis and proceed to a delay route for the duration of the break. Again, similar to the tool offline issues mentioned previously, the break item will not be able to engage with the operator if it is already performing a task, but will wait until it has finished conducting its current task. This is a consequence of the modelling strategy, but not an unrealistic scenario assuming that operators must fulfil their current task before taking breaks.

5.6.5 Rework and scrap

Two scrap checks are employed and tested after each operation step. The first check, samples a probability that the full lot is scrapped. If this test fails, the lot is recorded

and exited from the simulation model. If the test is passed, the wafers in each lot are then tested against a sample of the wafer scrap probability. Any failed wafers are removed from the lot, recorded and exit the simulation. The remaining wafers in the lot continue onto the next step in the lot's process route.

The mechanisms that check for rework are very similar. Firstly the lot is checked to see if it needs to be entirely reworked. If it fails this check, then it departs from the normal process route and joins a rework route according the rework sequence file. Once the reworked steps are completed the lot will rejoin its original process route.

On the other hand, if the lot passes the rework check, then the individual wafers that make up the lot are tested against a wafer rework probability. The wafers that fail this test are separated and a new rework lot is formed that takes the failed wafers through a number of rework steps according to the rework sequence file. Once these rework steps have been complete the rework lot merges back onto its main process route and continues.

Due to a lack of information in the specification, it was unclear whether reworked wafers, in a newly created rework lot, should merge back with their original lot once they have executed their rework steps. The simulation results showed that given that the probability of reworking at least one wafer per operation step was quite high, holding back lots for their reworked wafers to 'catch up' had a very negative impact on the model cycle time. Therefore, it was assumed that the original lot consisting of wafers that passed the rework test could continue and with a reduced wafer count.

5.6.6 Capturing the model output

The model uses a number of custom blocks to write timestamps to an output database stored in ExtendSim. There are three tables where information is written; *LotTrace*, *ToolTrace*, and *OperatorTrace*. Each time a lot, tool or operator completes a task, a row is recorded in the appropriate database table given in Tables 5.6-5.8.

Table 5.6: Description of *LotTrace* database for collecting model output.

Field	Description
useRow	Indicates if the row in the database is being used (for faster writing).
LotID	The unique lot ID, given when the lot entered the system.
RWFlag	Indicates whether the current step is a rework step.
ProductID	The lots product ID.
ProcessFlow	The lots process flow ID or rework sequence if it is rework.
StepID	The lots step ID or rework sequence step if it is rework.
WaferQty	The current wafer quantity of the lot at that point.
ToolsetID	The toolset required to perform the current step.
OperatorSetID	The operator required to perform the current step on the designated tool.
ArrivalTime	The point of arrival of the lot to the current process or step.
Batched	The timestamp immediately after the lot has formed a batch (if required).
Batch Identifier	The ID given to lots of the batch, if zero then no batching.
Paired with Tool	The timestamp placed after the lot/batch has captured an appropriate tool.
Finished Tool Setup	The timestamp recorded after the captured tool has been setup (if needed) for the current step.
Paired with Loading Op	Timestamp recorded after the lot/tool pair has captured an appropriate operator for loading.
Tool Loaded	Timestamp recorded after the lot has been successfully loaded onto the tool.
Paired with Process Op	Timestamp recorded after the lot/tool pair has captured an appropriate operator for processing.
Finished Processing	Timestamp recorded after the lot has been successfully processed.
Paired with Unload Op	Timestamp recorded after the lot/tool pair has captured an appropriate operator for unloading.
Unloaded	Timestamp recorded after the lot has been successfully unloaded from the tool.
Wafer Travel Finished	Timestamp recorded after the lot has completed its travel time component of processing.
ScrapFlag	Was the lot fully or partially scrapped.
Num Wafer Scrapped	How many wafers were scrapped from the lot.
CurrentRWFlag	Is this step part of a rework return.
RWwaitID	The ID so that the reworked child lot can merge with its parent lot.
RWDelayTime	The timestamp recorded immediately after the reworked child lot merged with its waiting parent lot.
Paired with Transport Op	Timestamp recorded after the lot has captured an appropriate operator for transporting.
Lot Finished Transporting	Timestamp recorded after the lot has finished transporting and has arrived at its subsequent step or exited the system if it was on its final step.

Table 5.7: Description of *ToolTrace* database for collecting model output from the tools.

Field	Description
useRow	Indicates if the row in the database is being used (for faster writing).
loopTypeFlag	Indicates if this row is recording a processing, maintenance or repair event.
ToolID	The tools ID
ToolsetID	The toolset ID that it belongs to.
UniqueToolID	The unique ID within its toolset.
Start Time	When the tool becomes available.
Paired with Lot	When a lot captured/reserved the tool.
Finished Lot	When the lot releases the tool.
Paired with DT	Timestamp to indicate the beginning of a downtime event.
Got DT Operator	Timestamp to record the time the repair operator got to the downed tool.
Finished DT Event	Timestamp to indicate when the repair finished.
Got Maintenance Op	Timestamp to record the time the repair operator got to the tool waiting for a maintenance task.
Finished Maintenance	timestamp to indicate when the maintenance finished.
Repair OperatorSet ID	The operator set used to perform the maintenance/repair.

Table 5.8: Description of *OperatorTrace* database for collecting model output from the operators.

Field	Description
useRow	Indicates if the row in the database is being used (for faster writing).
loopTypeFlag	Indicates if this row is recording a loading, processing, unloading, operator break, maintenance or repair event.
OperatorSetID	The operator set ID
UniqueOperatorID	The unique operator Id within its set.
StartTime	The time when the operator became available.
Got Lot/Tool/DT/Break	Time when the operator engaged and paired with a mating item.
Finished Lot/Tool/DT/Break	Time when the operator unpaired with the mating item.

5.6.7 Checking model stability

During the initial pilot run the program outputs a cycle time trace of the lots. This is necessary so that the user can confirm that the simulation is not unbalanced and will converge to a solution. The statistical process control (SPC) method described in Section 3.4.1 prevents the program considering any output from an unbalanced model. However, if the model was overloaded, the SPC algorithm would find that it is not in steady state and double the model run time. Hence the system of checking the cycle time trace output from a pilot run confirmed to the user that the loading level is not causing

the model to become unstable. Figure 5.5 shows a sample output from the application, running dataset 1 at its prescribed loading level. It can be seen that the model is stable and cycle time is not increasing in an unstable fashion.

It is worth noting that this safety check is not asking or requiring the user to confirm that steady state has been achieved, this is tested using the SPC method. This stability check is merely in place to prevent the waste of computational resources and ensure the user is aware if the model has become overloaded and the output is unstable.

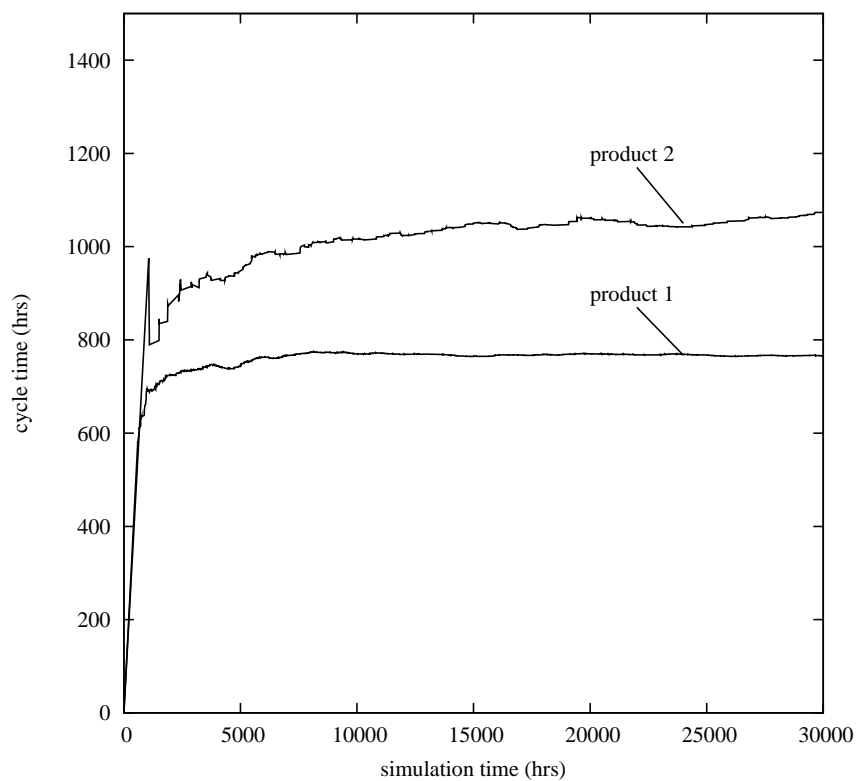


Figure 5.5: Cycle time trace of both products in Sematech dataset 1.

Once confirmed, the programme constructs the operating curve by capturing the performance measure (cycle time) for a number of replications at each design point on the operating curve.

5.7 Analysis of Sematech Dataset 1

This section examines the operating curves output from dataset 1 and compares the curves of various configuration strategies to show how this tool can be used to identify any potential weaknesses in the system or large contributors to cycle time. Table 5.9 describes the configuration of dataset 1.

Table 5.9: Description of Sematech dataset 1 from *MASM Lab Factory Datasets* (1996).

Type of product	Non-volatile memory
Number of process flows in dataset	2
Number of different products [†]	2 (1 per process)
Dataset products make up what % of factory	95% - 98%
Average number of process steps per mask layer	15
Are operators modelled?	Yes
Is rework modelled?	Yes
Number of equipment groups (toolsets)	83
Approximate wafer starts per month	16,000
Raw process time	Product 1 = 313.4 hrs Product 2 = 358.6 hrs

[†] Note: more products in real factory.

There are two main product flows within dataset 1, namely Product 1 and Product 2, of which there is a 2:1 product ratio. Both products make up 95% of the overall factory products. Operators and rework are included in the dataset, as well as scrap. Operator breaks are not included but the machines are unreliable with the majority having at least one downtime stochastic pattern associated with it. The average availability of the machines in the dataset was calculated as 0.917 using Eq.(2.10) on pg. 29. No operators are required to repair the machines in dataset 1.

The configuration file accompanying the dataset also gives some brief results attained from a test run conducted on a simulator known as *Factory Explorer*. Table 5.10 lists the results. The average cycle time for Products 1 and 2 were 701.84 and 907.02 hours respectively. Setup avoidance measures were carried out the `Implant` workstation, however, there is no mention of operators being modelled.

Defining the system utilisation can be difficult and it is easier to use a measure such as

Table 5.10: Sample run of Sematech dataset 1 using Factory Explorer.

Input rate	95% of maximum possible input rate
Distribution of process times	constant
Distribution of times between random failures	exponential
Distribution of time between PM's	n/a
Distribution of PM durations	n/a
Distribution of operator breaks	n/a
Dispatch rules followed	FIFO
Lot release policy	setup avoidance at TS10 & TS11 (Implant)
Language/simulator used	constant time between lot releases
Run length	Factory Explorer
Replications	50,000 hrs
Truncation of initial output (warm-up)	1
Average cycle times by product	10,000 hours
	Product 1 = 701.84 hrs
	Product 2 = 907.02 hrs
Average number of outs	Product 1 = 12,650 lots
	Product 2 = 6,198 lots

starts rate. Thus the operating curve output from the application is a plot of the x-factor against the starts rate of the system. The starts rate is measured in lots per hour and the x-factor can be determined from the ratio of average cycle time to the weighted average of the raw process time per product, as in Eq.(5.7), where π_i is the proportional product mix.

$$t_{0 \text{ (fab)}} = \sum_{i=1}^n \pi_i t_{0_i}, \quad \text{for } i = 1, 2, 3, \dots, n \quad (5.7)$$

5.7.1 Batch size policies

Changing the batch size policy had a large effect on the resulting average cycle time of lots in the system. The time taken for lots to form batches is a direct and heavy contributor to cycle time for the fab configuration described by dataset 1. Tables 5.11 and 5.12 show the model cycle time and x-factor results under a maximum and minimum batch sizing policy respectively. Figure 5.6 plots the two operating curves for each batch size policy. This result indicates that cycle time is negatively impacted at low loads, even when a minimum batch size policy is in force.

Increasing the system loading and assuming a minimum batch size policy, the average

Table 5.11: Simulation model results for Sematech dataset 1 with a maximum batch size policy and no operators, downtime or rework.

capacity	starts rate (lots per day)	run length (hrs)	warm-up (hrs)	bottleneck	u_{BN}	CT (hrs)	x-factor
0.1	1.19	50,000	10,000	TS67	0.1029	653.88	1.946
0.2	2.38	50,000	15,000	TS67	0.1998	508.77	1.514
0.3	3.57	50,000	15,000	TS67	0.3041	468.17	1.393
0.4	4.76	50,000	10,000	TS67	0.4025	451.25	1.343
0.5	5.95	50,000	10,000	TS67	0.5032	443.97	1.321
0.6	7.14	50,000	10,000	TS67	0.6036	443.45	1.320
0.7	8.33	50,000	20,000	TS67	0.7014	450.14	1.340
0.8	9.52	50,000	25,000	TS67	0.8011	461.91	1.375
0.85	10.12	50,000	12,500	TS67	0.8475	470.75	1.401
0.9	10.71	50,000	12,500	TS67	0.8927	485.34	1.444
0.95	11.31	50,000	20,000	TS67	0.9473	531.31	1.581
1.00*	11.90	100,000	n/a	TS67	0.9986	1044.92	3.111

* The simulation could not achieve steady state at this design point.

Table 5.12: Simulation model results for Sematech dataset 1 with a minimum batch size policy and no operators, downtime or rework.

capacity	Starts rate (lots per day)	run length (hrs)	warm-up (hrs)	bottleneck	u_{BN}	CT (hrs)	x-factor
0.1	1.19	100,00	30,000	TS30	0.1187	513.66	1.529
0.2	2.38	50,000	15,000	TS30	0.2401	431.09	1.283
0.3	3.57	50,000	20,000	TS30	0.3683	407.92	1.214
0.4	4.76	50,000	10,000	TS30	0.4794	397.87	1.184
0.5	5.95	50,000	10,000	TS30	0.6045	394.42	1.174
0.6	7.14	50,000	12,500	TS30	0.7234	396.23	1.179
0.7	8.33	50,000	20,000	TS30	0.8411	404.89	1.205
0.8	9.52	70,000	30,000	TS30	0.9470	434.60	1.294
0.85 [†]	10.12	70,000	n/a	TS30	0.9988	1136.91	3.384

[†] The simulation could not achieve steady state at this design point.

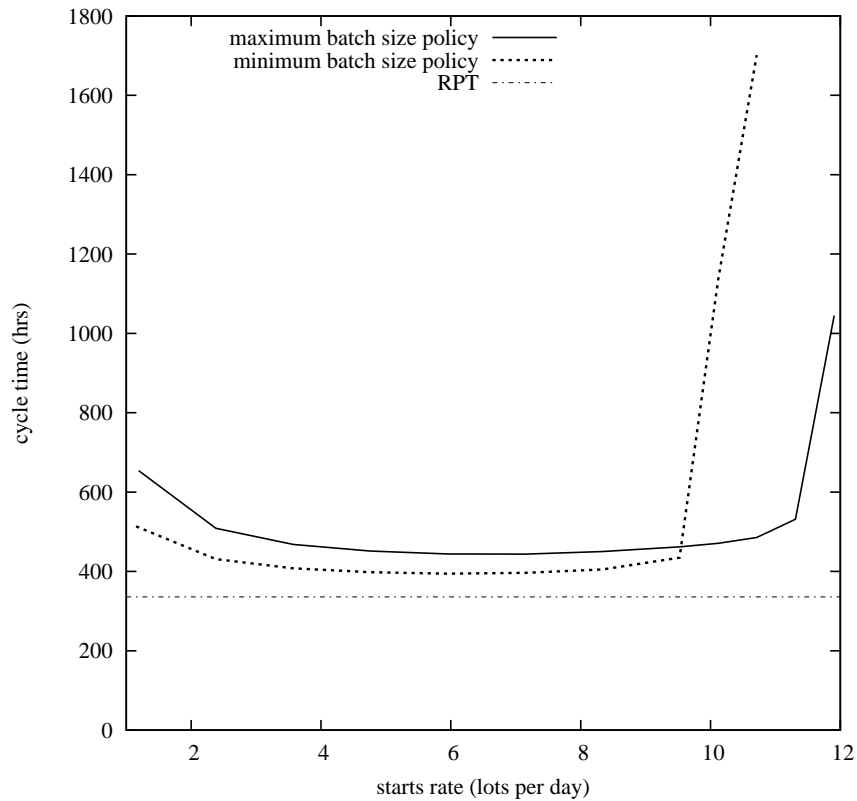


Figure 5.6: Operating curves for dataset 1 using a minimum and maximum batch sizing policy.

cycle time does not increase significantly until the starts rate is greater than approximately 9.5 lots per day. The next design point and any loading level above this failed to achieve steady state on the simulation model. Therefore, the values for cycle time and x-factor at this level should be noted as unstable, and likely to be higher if the fab is loaded at these high levels for a long period of time.

Neither batch size policy produced a model that could attain cycle time approaching the raw process time, even at very low loads, therefore, ‘waiting to batch time’ is a large contributor to cycle time. It appears that the best batch size policy (given that only two policies were examined) is to encourage a minimum batch sizing until the starts rate increases above approximately 9.52 lots per day. At this point the fab becomes more efficient if it transfers to a maximum batch size policy.

5.7.2 System bottleneck analysis

There are a number of bottlenecks within the system that switch dominance depending on the particular configuration. Using the results in Tables 5.11 and 5.12, the bottleneck was found to be at toolset 30 (known as DRIVE OX) for a minimum batch size policy and toolset 67, (MATRIX toolset), for a maximum batching size policy.

Toolset 67 consists of seven single wafer processing tools that perform six operations on Product 1 and eight operations on Product 2. The operations take on average 2.618 minutes per wafer which is approximately 2 hours per lot, assuming that most lots consist of just under 48 wafers. Using Eq.(5.8) it is possible to calculate the maximum arrival rate $r_{a (max)}$ that the toolset can accommodate for each utilisation level u .

$$r_{a (max)} = \frac{um}{t_e} \quad (5.8)$$

Given that $\frac{2}{3}$ of all lots in the system are of Product 1, that make six passes through toolset 67, and the remaining $\frac{1}{3}$ of lots in the system are of Product 2 and make eight passes through the bottleneck, then the average number of visits to the bottleneck toolset for any lot is 6.66 during its process route. This can be used to calculate the average maximum arrival rate for the system, given that the bottleneck toolset is the limiting factor.

Bottleneck analysis of dataset 1 using a maximum batch size policy

Table 5.13 shows a summary of maximum allowable system arrival rate according to this calculation and compares it with the actual system arrival rates used for the simulation model. They show a close correlation with an average difference of about 5%. As expected the actual arrival rates used are strictly never greater than the maximum possible toolset arrival rate to maintain the appropriate system utilisation.

This shows that in this particular dataset, (without employing operators or downtime in the system), that toolset 67 is an appropriate representation of overall system capacity

Table 5.13: The approximate average arrival rate permissible for the system and for the bottleneck toolset TS67, given that $m = 7$.

u	r_a (bottleneck)	th_a (system)		
		calculated	actual	difference
0.1	0.35	0.053	0.049	7.5%
0.2	0.7	0.105	0.099	5.7 %
0.3	1.05	0.158	0.149	5.7%
0.4	1.4	0.210	0.198	0.9%
0.5	1.75	0.262	0.248	1.5%
0.6	2.1	0.315	0.297	5.7%
0.7	2.45	0.367	0.347	5.5%
0.8	2.8	0.420	0.397	5.5%
0.9	3.15	0.472	0.440	6.7%
1.0	3.5	0.525	0.496	5.5%

when using a maximum batch sizing policy. Using an M/D/m cycle time approximation of all operations that use this toolset, it is possible to compare the operating curve for the bottleneck toolset with that of the simulated curve for the fab (assuming a maximum batching policy). Use of the M/D/m queueing model is justified because there are many independent arrival sources (> 14) to TS67 (see Section 4.3.1 for discussion). Also, the process pattern is deterministic, as the model does not put a distribution around process time. Figure 5.7 plots the M/D/m approximation for TS67 and the simulated operating curve for the system under a maximum batch size capacity using Eq.(5.9) where $m = 7$ and the weighted average raw process time (RPT) is calculated using Table 5.14,

$$CT_{M/D/m} = t_0 \left(1 + \frac{u\sqrt{2(m+1)-1}}{m(1-u)} \right) \quad (5.9)$$

Bottleneck analysis of dataset 1 using a minimum batch size policy

Toolset 30 (DRIVE OX), a batching toolset with a capacity of two tools, is the bottleneck for a minimum batch size policy, particularly as the toolsets' minimum batch capacity is two lots with a very large process times of 22.5 hours. TS30 only performs one operation on one of the two products; Product 2, which makes up only a third of all products in the fab. This means that any increase of the product ratio in favour of Product 2 could

Table 5.14: Operation details for TS67.

Product ID	Mix	Operation ID	Operation Name	Flow Time (hrs)
1	2	111	2GATE ETCH.8	2.144
		135	N_STRIP_1	2.144
		145	P_STRIP_1	2.144
		170	C IMP STRIP_1	2.144
		187	METAL ETCH_3	2.144
		200	PAD ETCH_4	2.144
2	1	123	SILI ETCH_6	2.144
		138	ARRAY STRIP_1	2.144
		165	N_STRIP_1	2.144
		175	P_STRIP_1	2.144
		195	CONT ETCH_3	1.686
		205	C IMP STRIP_1	2.144
		222	METAL ETCH_3	2.144
		235	PAD ETCH_4	2.144
weighted average raw process time				2.1211

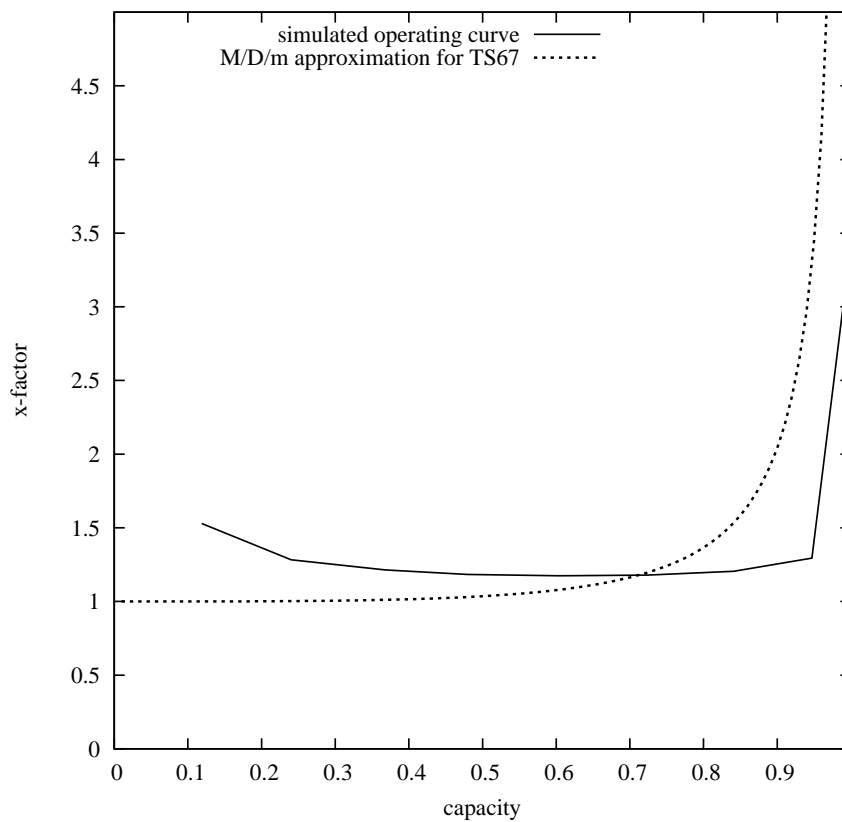


Figure 5.7: Comparison of M/D/m approximation for bottleneck toolset TS67 and fab operating curve predicted by simulation model.

have a significant impact on the operating curve.

Analysis of the maximum capacity of TS30 is easier than TS67, given that it only performs one operation type on one of the products. Given that its loading and unloading time is 0.5 and 0.13 hours respectively, then the maximum capacity for the toolset with a capacity of 2 and a batch size of 2 is 4.15 lots per day. Assuming a consistent product ratio of 2:1 then the maximum theoretical capacity for the system is 12.451 lots per day. In fact, it falls quite short of this according to Table 5.12, where anything above approximately 9.5 lots per day produces an unstable model.

However, there is caveat to employing a minimum batch size policy in the model. In the case of the maximum batch size policy, this is realistic, assuming that fab management want to maximise the utilisation of expensive or power hungry equipment. While, the minimum batch size policy employed in the model is less realistic. For example, in the real system, if three lots arrived to TS30 simultaneously, they would be batched and processed together. However, in the model, only two would be batched together and the third would have to await the arrival of a fourth lot before processing. This is a consequence of the modelling strategy employed; a model that uses an entity-centric modelling strategy, must give dominance to one or another entity, in a master-slave fashion. So, despite the fact that all three arrive at the toolset together in a discrete point in time, they actually arrive one after the other in terms of model code execution. Stepping through the series of simulation events; when the first lot (in the execution sequence) arrives it searches for another lot of same type, if it does not find one it waits for zero time-until the next lot in the sequence arrives. These two lots then form a batch (the master) assuming two lots is minimum batch size, and then enter the Pairing block to search for the appropriate tool entity (slave) to pair with. During these events, which have been all been carried out at the same discrete point in simulated time, the third lot arrives and must wait as it cannot form a batch.

5.7.3 Comparison with the CXFC approximation

Comparing the operating curves generated by the simulation model with those of the complete x-factor contribution (CXFC) approximation shows some key differences. Note that the CXFC values were not recalculated here but were taken from Delp et al. (2006) which is based on Eq.(2.8), given in Section 2.4.1. It is also assumed from this point that a maximum batch size policy is employed. The resulting plot is depicted in Fig. 5.8 and shows that above 10 lots per day, the model shows a steeper increase in the x-factor. Unfortunately, it was not possible to compare the low loading increase in x-factor because these results were not reported in the article. However, it is unlikely that the results produced by Delp et al. would match those of the simulation model, given that the x-factor does not account for the waiting to batch time which has such a large negative impact on the x-factor at the low loading level.

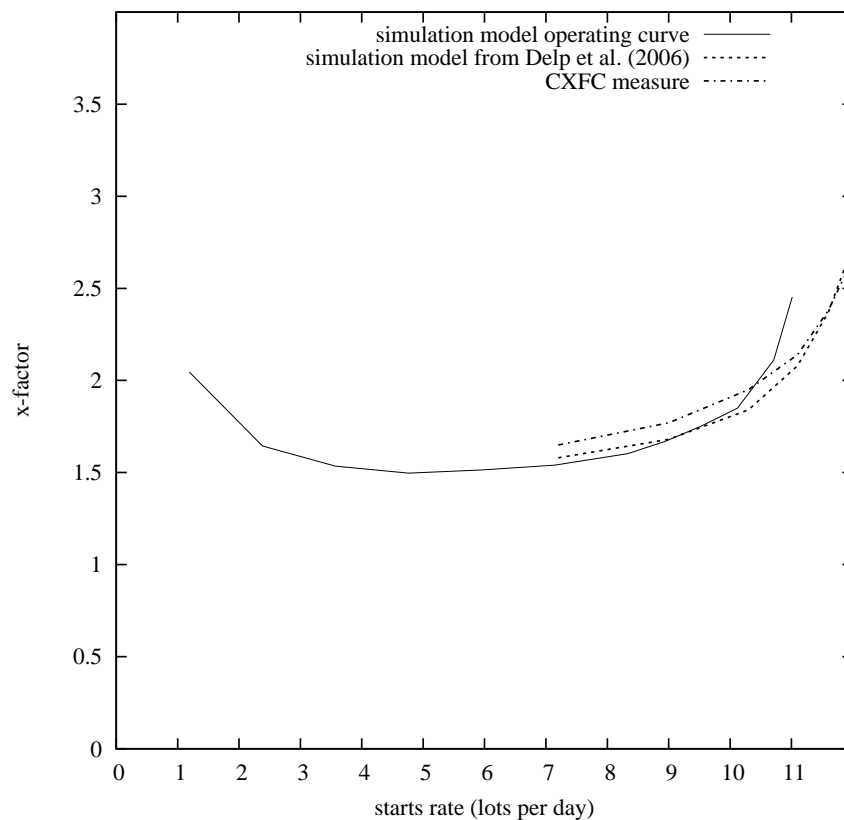


Figure 5.8: Comparison of results for flexible reusable model, x-factor and CXFC produced by Delp et al. (2006).

Table 5.15: Comparison of results for flexible reusable model, simulation model produced by Delp et al. (2006), and the CXFC approximation.

simulation model		Delp et al. (2006)		
starts rate (lots per day)	x-factor	starts rate (lots per day)	x-factor	CXFC
1.190	2.0458	7.2	1.58	1.65
2.381	1.6443	9.0	1.68	1.77
3.571	1.5345	10.3	1.84	1.95
4.762	1.4964	11.1	2.08	2.14
5.952	1.5141	11.6	2.37	2.38
7.143	1.5400	11.9	2.64	2.59
8.333	1.6027	12.0	2.93	2.68
8.929	1.6679			
9.524	1.7513			
10.119	1.8495			
10.712	2.1106			
11.012	2.4533			

5.7.4 Downtime

The toolsets described by dataset 1 are considered unreliable given that all have at least one maintenance/downtime stochastic mechanism. Figure 5.9 shows that the addition of unreliable machines causes the operating curve to shift upwards and increase the x-factor at a lower loading. The location of the bottleneck for the system remains at TS67, which has one time-based failure mechanism called `failure`, and has an availability A given by,

$$A = \frac{t_f}{t_f + t_r} = \frac{8.45}{153.766 + 8.45} = 0.948 \quad (5.10)$$

A list of the top ten toolsets with the lowest availability (most unreliable) is given in Table 5.16. Toolset 71, known as TEGAL has the lowest availability of 0.69. This toolset was found to be relatively lowly utilised, however, if its utilisation increased it may become the dominant bottleneck in the system. This is particularly true given that the toolset consists of only one machine and has very long MTTR of 66.82 hours. For such low availability toolset, (and other toolsets for that matter), it is always better to have shorter more frequent offline periods than less frequent and longer periods of

Table 5.16: Unreliable toolsets in dataset 1 ranked by least availability.

rank	toolset name	toolset ID	capacity	mechanism	m_f (hrs)	m_r (hrs)	A
1	TEGAL	71	1	time-based	152.07	66.82	0.69
2	GENUS	17	2	time-based	34.44	12.99	0.73
3	MED_CURRENT_IMP	10	4	time-based	23.37	8.05	0.74
4	VARIAN	15	2	time-based	30.33	9.41	0.76
5	PROMETRIX	22	1	time-based	48.00	12.00	0.80
6	LAMINATOR	23	1	time-based	48.00	12.00	0.80
7	DELAMINATOR	24	1	time-based	48.00	12.00	0.80
8	STRASBAUGH BACKGRIND	28	3	time-based	48.00	12.00	0.80
9	ANELVA	16	2	time-based	44.91	10.81	0.81
10	HIGH_CURRENT_IMP	11	4	time-based	29.85	6.93	0.81
-	-	-	-	-	-	-	-
18	DRIVE_OX [†]	30	2	time-based	103.34	11.39	0.90
59	MATRIX*	67	7	time-based	153.77	8.45	0.95

[†] bottleneck toolset when min. batch size policy is employed

* bottleneck toolset when max. batch size policy is employed

unavailability (Hopp and Spearman, 2001).

Table 5.17: Simulation model results for dataset 1 with unreliable machines.

capacity	starts rate (lots per day)	run length (hrs)	warm-up (hrs)	bottleneck	u_{BN}	CT (hrs)	x-factor
0.1	1.19	50,000	15,000	TS67	0.1069	687.21	2.046
0.2	2.38	50,000	15,000	TS67	0.2165	552.35	1.644
0.3	3.57	50,000	15,000	TS67	0.3109	515.47	1.534
0.4	4.76	50,000	10,000	TS67	0.4222	502.67	1.496
0.5	5.95	50,000	10,000	TS67	0.5228	508.62	1.514
0.6	7.14	50,000	10,000	TS67	0.6268	517.30	1.540
0.7	8.33	50,000	20,000	TS67	0.7344	538.36	1.603
0.75	8.93	50,000	25,000	TS67	0.7906	560.27	1.670
0.8	9.52	60,000	12,500	TS67	0.8457	588.27	1.751
0.85	10.12	60,000	12,500	TS67	0.8935	621.28	1.849
0.9	10.712	70,000	15,000	TS67	0.9412	708.98	2.111
0.925 [†]	11.01	70,000	n/a	TS67	0.9721	824.11	2.453

[†] The simulation did not achieve steady state.

5.7.5 Operators

Examining the operating curve of dataset 1 with operator requirements included in the simulation model shows that the system efficiency significantly reduces. By introducing

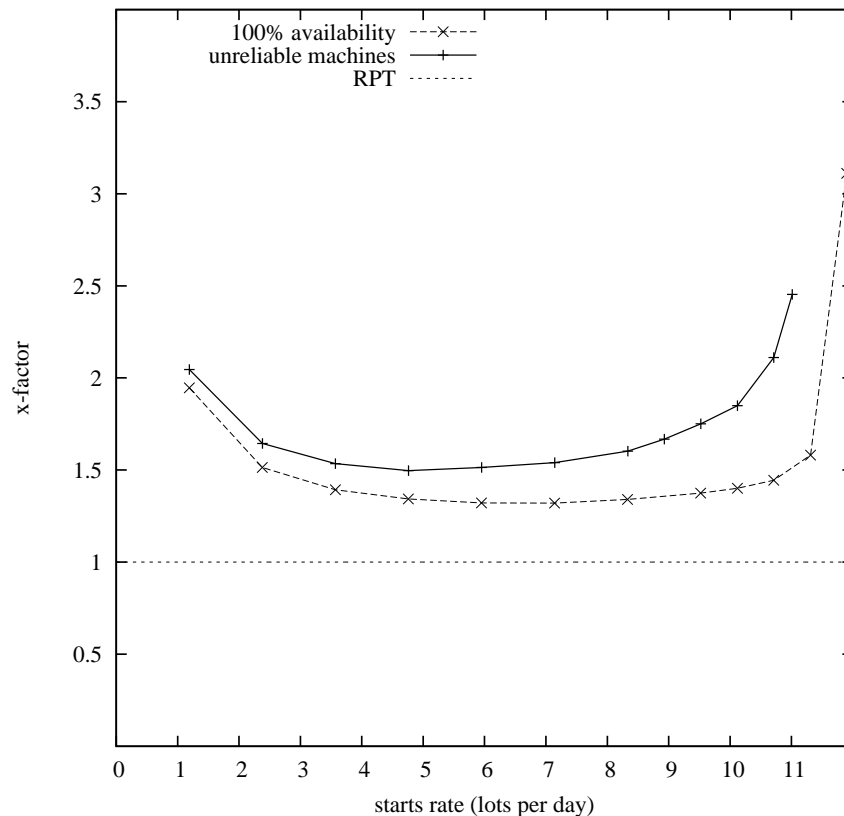


Figure 5.9: Comparison of operating curves for dataset 1 with unreliable machines according to Table 5.11 and Table 5.17.

operators, the bottleneck for the system moves from TS67 (under max. batch policy) and TS30 (under min. batch policy) to operator set no. 26 called the MATRIX_OP. This operator set has a capacity of just one operator whom is required by TS67, the machine bottleneck. Unsurprisingly then, the system now becomes restricted by the number of operators in this system, and the overall capacity of the toolset that the lots ‘see’ is just one given that an operator needs to both load the lot onto the tool and must process it. This means that it is likely that TS67 is a very manual process, and the capacity of the workstation effectively becomes the capacity of the operator set. Furthermore, if operator set OP26 consists of only one operator then bottleneck workstation is effectively offline when the operator is on a break.

These statements are verified by examining the operating curves in Fig. 5.10. The maximum starts rate with either a minimum or maximum batch size policy reduces from over 11 lots per day to approximately 1.6 lots per day which is about 14% of the rated

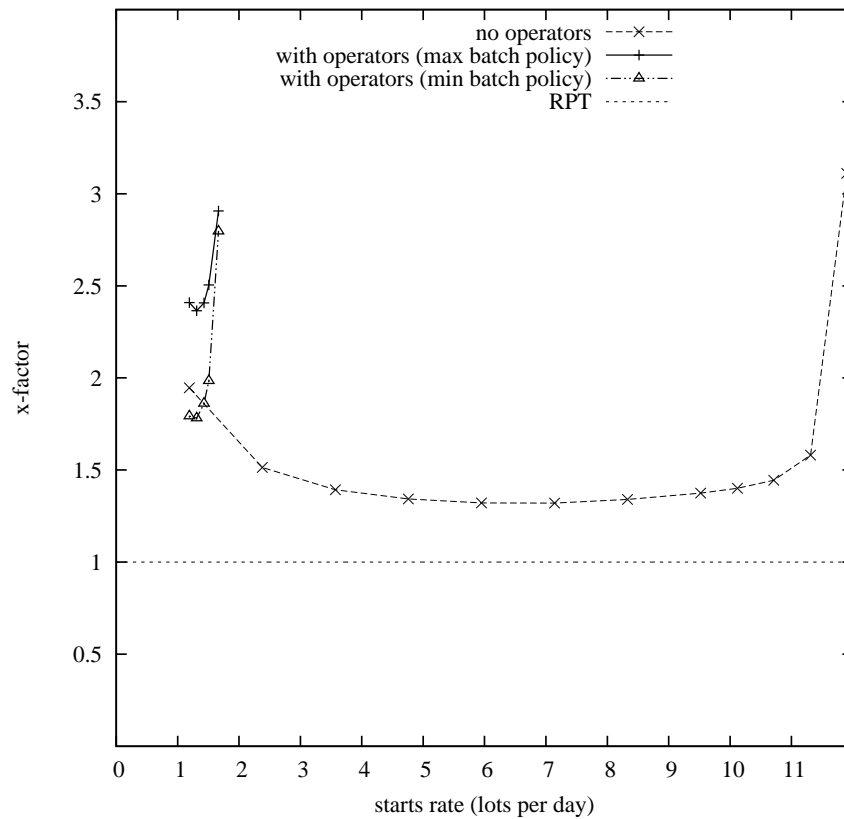


Figure 5.10: Operating curve for dataset 1 using operators under a minimum and maximum batching policy, according to Table 5.18.

capacity, according to Feigin et al. (1994). Although the system favours a minimum batch size policy here, the lack of operators allocated to the bottleneck workstation has a very large negative impact on the overall system efficiency. The resulting operating curves are high, shifted to the left, and sharply increasing for very small increases in the system loading.

5.8 Model Verification & Validation

Some of the verification techniques, summarised by Whitner and Balci (1989) and listed in Appendix G.1, were applied and the results are discussed in Table 5.19. Many of the custom blocks used for the FTM application described in Chapter 4 and included in Appendix B.4 were reused (subject to minor alterations). The verification results for these blocks can be found in Section 4.7.

Table 5.18: Simulation model results for dataset 1 using operators with a minimum and maximum batch sizing policy.

batching	capacity	lot starts (per day)	u_{BN}	OP_{BN}	run length (hrs)	steady state (hrs)	cycle time (hrs)	x-factor
max	0.1	1.1904	0.69	26	50,000	18,000	809.19	2.4089
	0.11	1.3095	0.75	26	50,000	15,000	794.33	2.3647
	0.12	1.4285	0.83	26	50,000	25,000	808.63	2.4073
	0.13	1.5079	0.89	26	50,000	25,000	841.33	2.5046
	0.14	1.6666	0.95	26	70,000	25,000	976.51	2.9070
min	0.1	1.1904	0.68	26	50,000	15,000	602.10	1.7922
	0.11	1.3095	0.77	26	50,000	14,000	598.34	1.7812
	0.12	1.4285	0.83	26	50,000	20,000	625.10	1.8610
	0.13	1.5079	0.88	26	50,000	25,000	666.60	1.9844
	0.14	1.6666	0.96	26	70,000	25,000	939.26	2.7961

Table 5.19: Techniques used to verify the ExtendSim model.

Type	Technique	Results and Comments
Informal	Walk-through	The structured walk-through was more heavily used as a validation tool but used little to verify the program.
	Code inspection	The source code was reviewed and inspected during model build and after the final version. Many of the errors in model execution occurred as a result of inconsistencies within the datasets. Therefore, it was necessary to clean up many of the datasets such that they followed the specification correctly.
Static	Syntax analysis	Modern software compilers ensured that the model syntax was complete and verified. The application code was compiled using Visual Basic for Applications (VBA) in Microsoft Excel. The ModL code used to create the custom blocks and model was verified using ExtendSim internal code compiler. Both compilers have an auto-compile feature which ensured that the syntax was correct during the programming phases.
	Structural analysis	Structural analysis was performed according to the best practices of coding. Plenty of comments were included with the code to ensure a clear and unambiguous meaning.
Dynamic	Top-down testing Bottom-up testing Black-box testing	Top-down, Bottom-up testing was applied using a black-box style that checked each model hierarchy with dummy entities to ensure each portion was consistent and did not produce erroneous outputs.
	Stress testing	Lot arrival rates were gradually increased until the model became overloaded and unable to reach steady state. MTBF and MTTR values were increased gradually causing very small tool availability. This resulted in very long lot queueing times in the model as expected. Other parameters such as scrap and rework probabilities were increased to verify that flow became increasingly restricted in the model.

continued on next page

continued from last page

Debugging	Debugging was an ongoing process during the model build and each version of the model and application was subjected to rigorous debugging.
Execution tracing	Data flow analysis and model tracing was performed for the simulation model by introducing only one lot into the system and monitoring its attributes as they changed during runtime. The result was that the lot experienced no queueing and minimal average process time and that the model data flow was correct. A zero-batching policy was employed during this process.
Execution monitoring	Trace animation capabilities in ExtendSim were used to validate the entity flow during runtime. Each entity in the model was given a different icon depending on its pairing with other entities or the process it was undergoing. This helped to ensure that all entities in the model were operating as required.
Regression testing	Regression testing (repeating the above procedures) was performed after each new model version.

All of the datasets contain sample run data, although the configuration of the sample runs is unclear in the documentation. Table 5.20 shows a comparison between the documented RPT for both products in dataset 1 and the results obtained from the simulation model described here. The RPT was calculated by summing the individual process times for each operation step along the process route for each product. The simulated RPT was greater than both the calculated and the documented RPT as a result of the high number of setups needed at each toolset due to the model beginning from an unknown state.

Table 5.20: Comparison of reported, calculated and simulated raw process times for Products 1 and 2 for dataset 1.

Product	Sematech documentation (hrs)	calculated (hrs)	simulated (hrs)
Prod 1	313.4	315.93	347.8
Prod 2	358.6	361.52	397.2

Further validation was performed by comparing the cycle time results of the simulation model with those from the configuration files in the dataset. Given that the factory usually produced 16,000 wafers per month, and assuming a month is made up of 30 days, the equivalent number of lot starts per day is 11.11, based on a lot of 48 wafers. If the test runs reported in the configuration files are based on running the Factory Explorer model

at 95%, this equates to approximately 10.55 lot starts per day and a cycle time of 701.84 and 907.82 hours for Product 1 and 2 respectively (see Table 5.10). If the ratio of Product 1 to 2 is 2:1, the weighted average cycle time is calculated as 770.5 hrs. This compares reasonably well (8% difference) to an average cycle time value of 708.98 hours reported by the simulation model results in Table 5.17. Again however, there is no indication in the configuration files as to the exact make-up of the simulation, i.e., whether operators and rework are included, or the type of batching policies that are implemented.

A number of other validation techniques were also applied. Table 5.21 shows the results of the validation tests that were performed on the model. An explanation of each test can be found in Appendix G.2.

Table 5.21: Techniques used to validate the ExtendSim model and application.

Technique	Results and Comments
Animation	ExtendSim's animation options were used extensively to validate the model.
Comparison to other models	A model built using SimPy in Python was used to validate the model. A detailed description of this model is given in Chapter 6.
Degenerate tests	Degenerate tests were performed by overloading the model by gradually increasing the arrival rate of test lots into the model and forcing utilisation of the toolsets to capacity, thus making the model unstable and unable to attain steady state. This behaviour was expected and helped in part to validate the model and examine the boundaries of operation.
Event validity	Event validity was ensured by monitoring the routing/events of entities in the model and comparing them against the dataset information. A list of the operations performed by each lot that was written to the ExtendSim database was monitored to ensure that each lot underwent its prescribed process route.
Internal validity	Very few replications according to the calculations described in Section 3.2 were required, meaning that the output results variability of the performance metric (cycle time) was very low.

5.9 IDEF Model Diagrams

Figs 5.11-5.20 include a partial IDEF0 representation of the ExtendSim model taken from the view of the modeller. The inputs and outputs include the flow of entities; lots, batches, tools, downtimes, operators and breaks. The controls consist mainly of entity attributes, and the mechanisms include the ExtendSim blocks used and their library. The

library *custom.lib* contains all of the custom blocks (see Appendix B.4) that were written for the model. Some of the more complicated sections of the model have been excluded from the IDEF0 interpretation in an effort to show clarity rather than completeness.

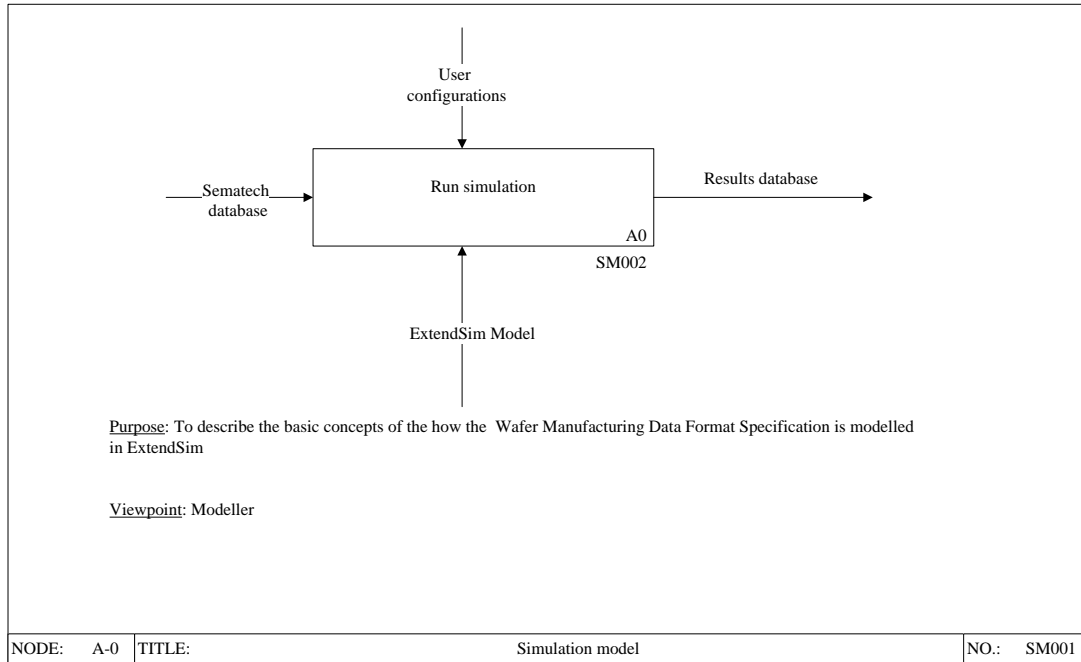


Figure 5.11: Overview IDEF0 diagram (A-0) for ExtendSim model.

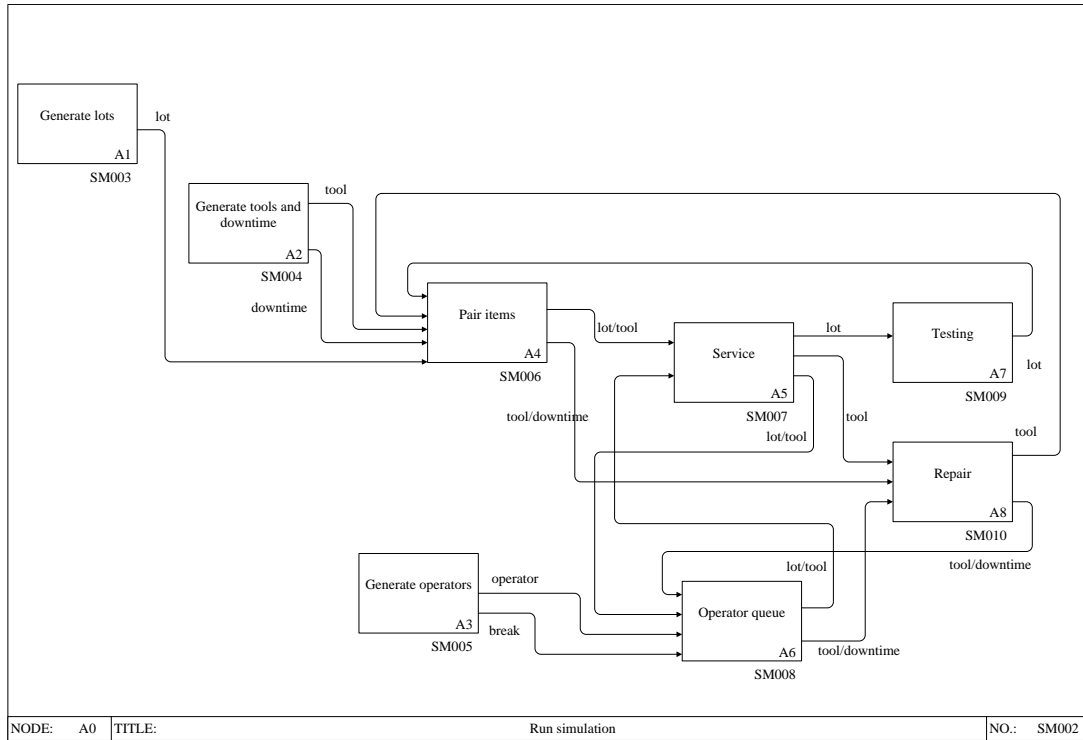


Figure 5.12: 'Run simulation' (A0) IDEF0 diagram for ExtendSim model.

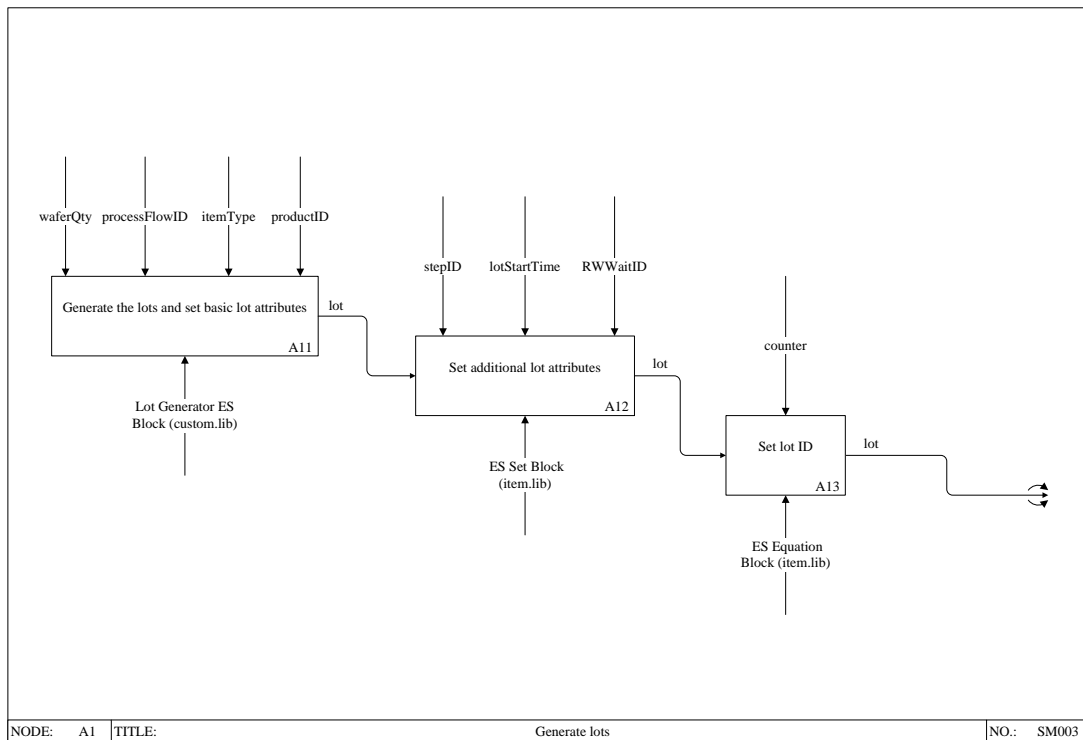


Figure 5.13: 'Generate lots' (A1) IDEF0 diagram for ExtendSim model.

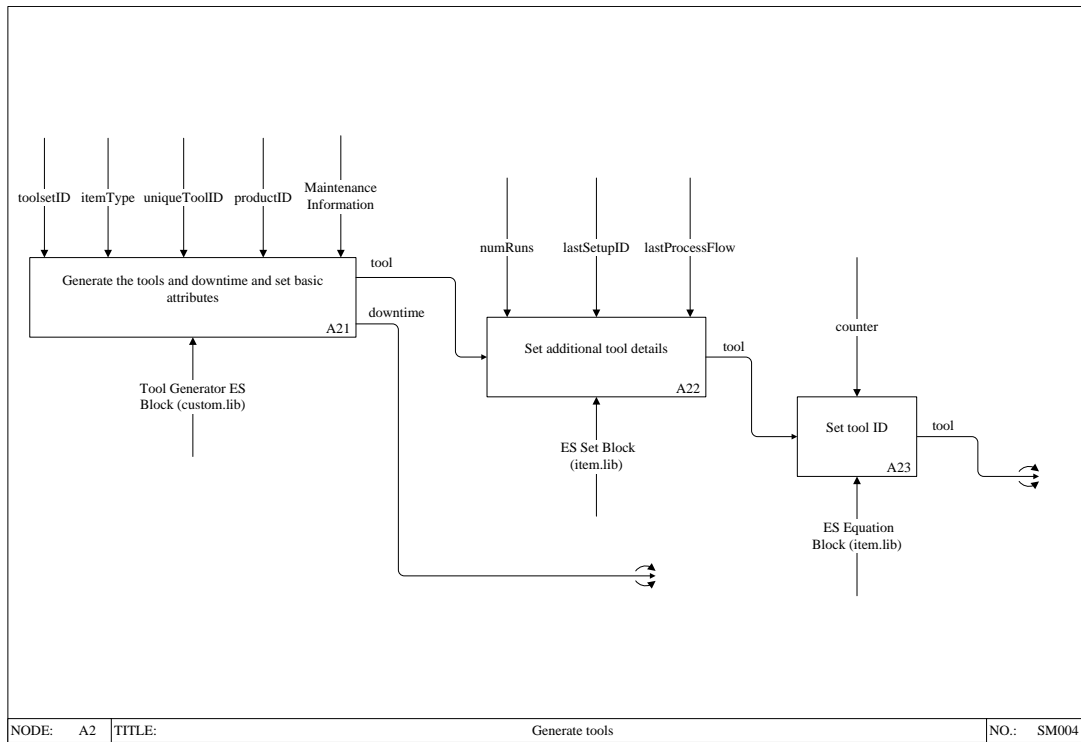


Figure 5.14: 'Generate tools' (A2) IDEF0 diagram for ExtendSim model.

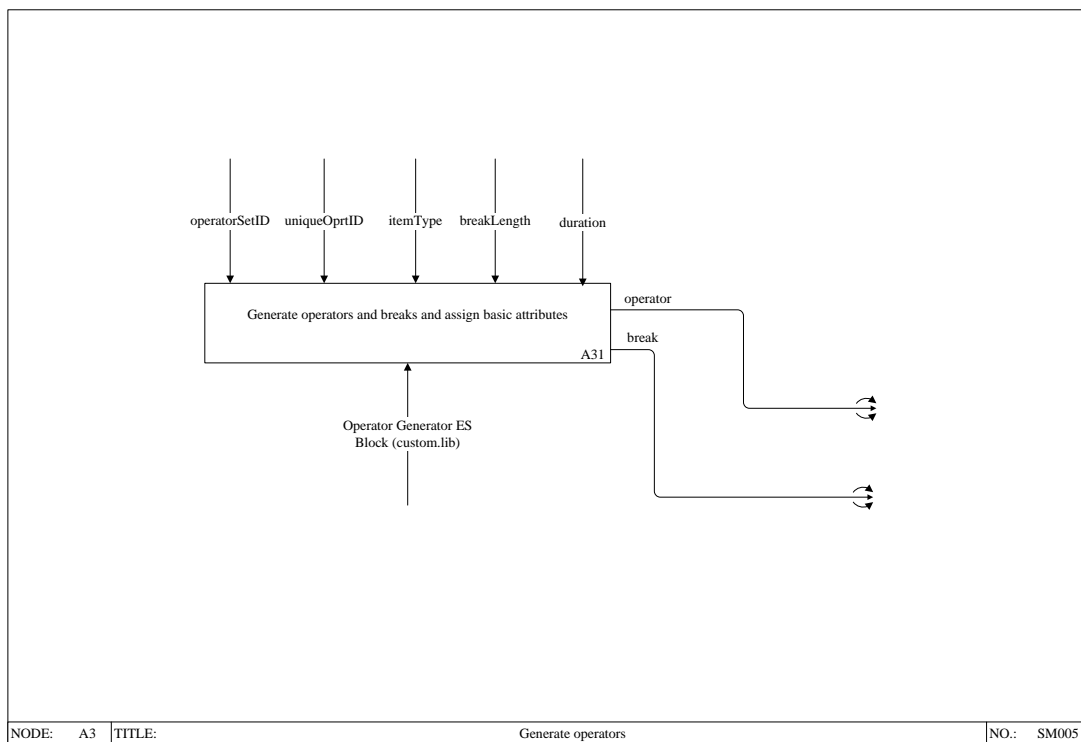


Figure 5.15: 'Generate operators' (A3) IDEF0 diagram for ExtendSim model.

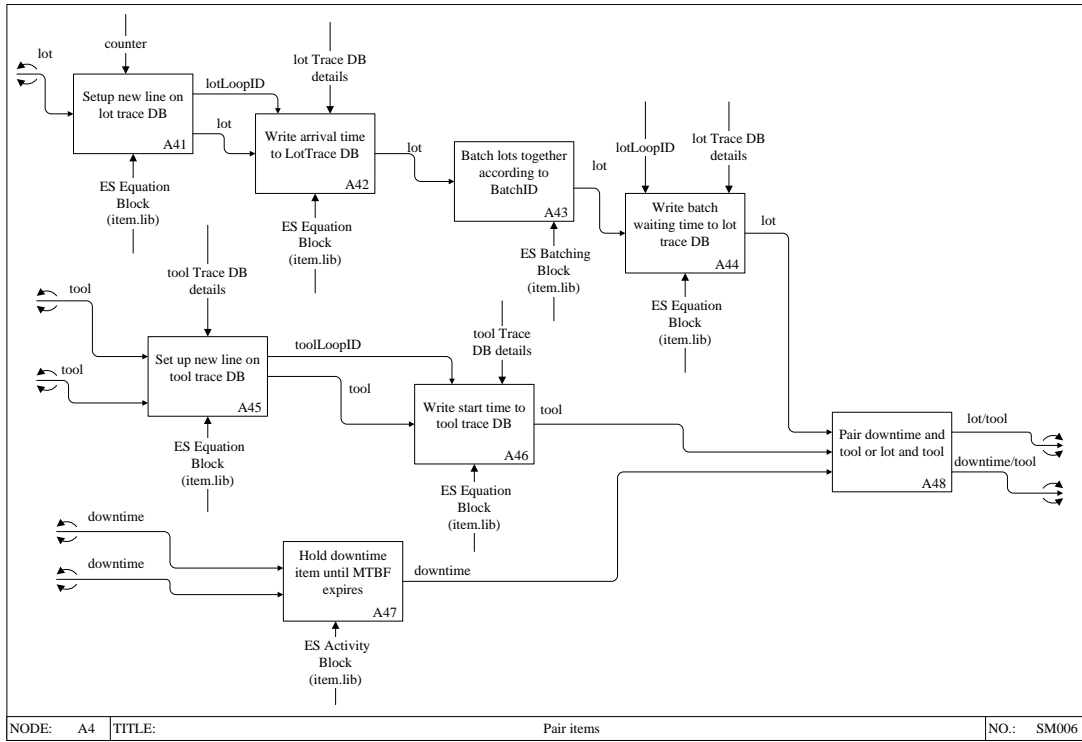


Figure 5.16: 'Pair items' (A4) IDEF0 diagram for ExtendSim model.

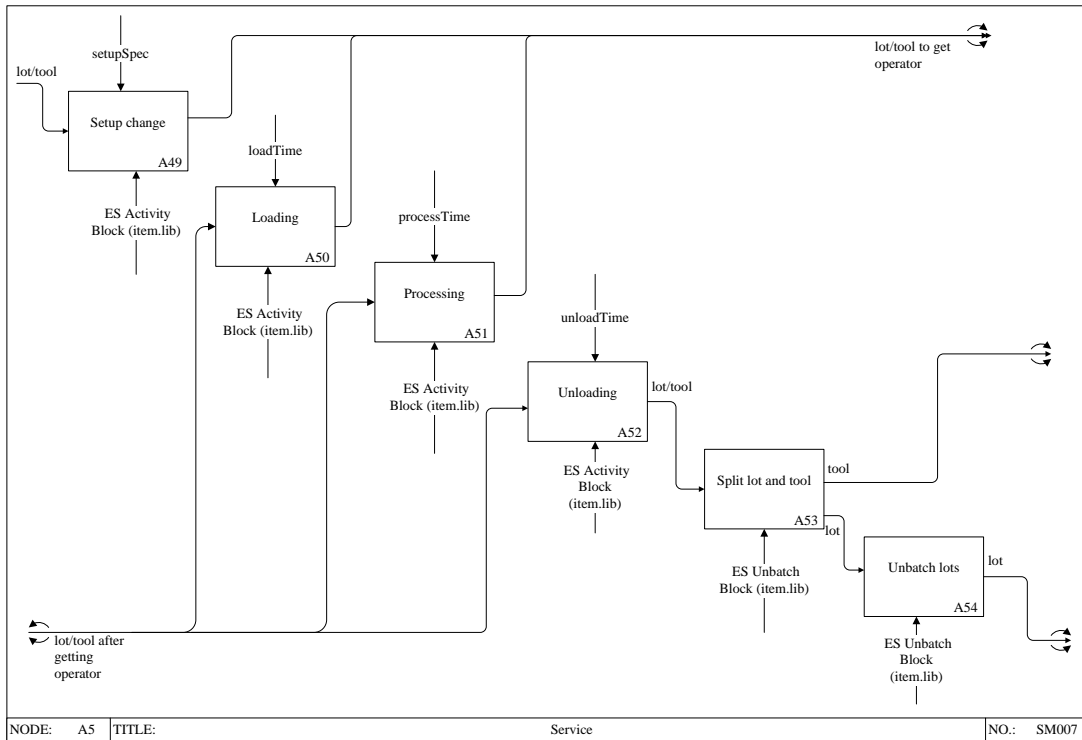


Figure 5.17: 'Service' (A5) IDEF0 diagram for ExtendSim model.

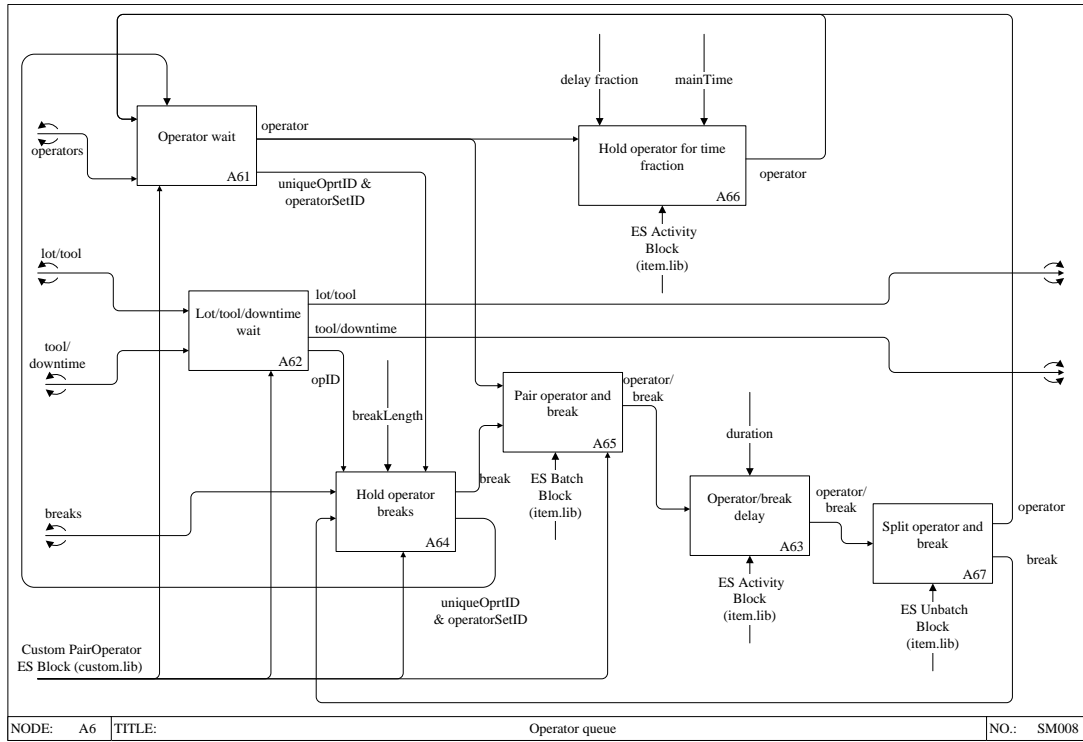


Figure 5.18: 'Operator queue' (A6) IDEF0 diagram for ExtendSim model.

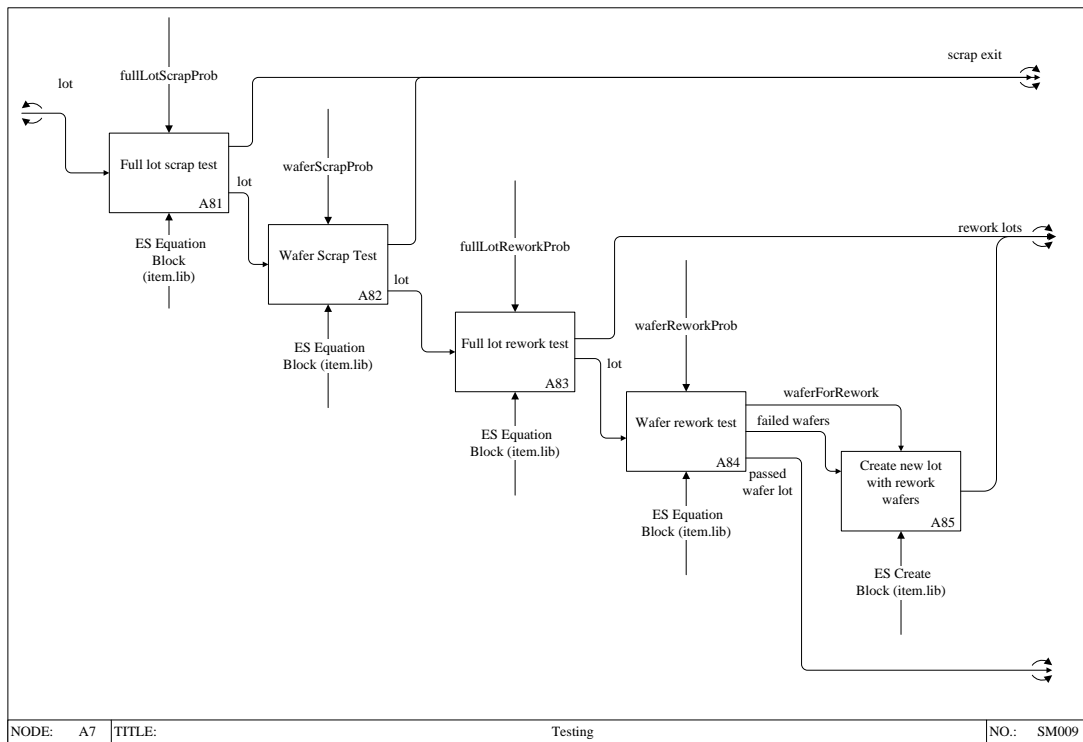


Figure 5.19: 'Testing' (A7) IDEF0 diagram for ExtendSim model.

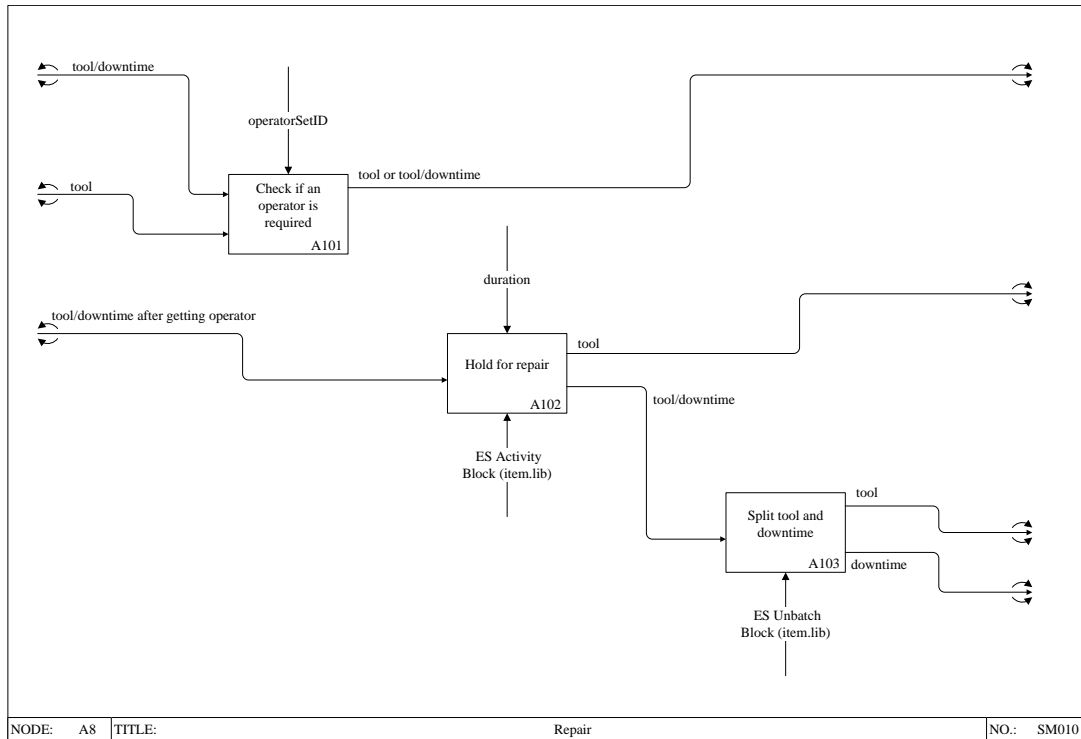


Figure 5.20: 'Repair' (A8) IDEF0 diagram for ExtendSim model.

5.10 Summary

Overall it was shown that a simulation model based on a flexible-reusable modelling strategy performed quite well when generating operating curves for various configurations of dataset 1. It was shown that the batching policy used had a strong influence on the efficiency of the system and helped to recommend particular policies in different loading ranges. It was also possible to examine the impact of unreliable machines on the factory operating curve which showed how changing the factory configurations caused a shift in the bottleneck. Comparing the operating curves generated by the model and those generated by queueing approximations showed that the simulation based curves outperformed the queueing approximations, which failed to account for the wait to batch time, setups, rework and re-entrancy in the system.

In the majority of cases, the datasets tested showed many errors and despite the *Semiconductor Wafer Manufacturing Format Specification* showing promise as a holistic descriptor for a semiconductor fab, there were many ambiguities over how the data should

be interpreted. Most notably, there is no information about variability. Furthermore, many of the datasets interpreted the specification differently and most required some pre-screening or adjustments before being used.

The robustness of the model was proved by performing multiple runs of various different configurations, and an extensive verification and validation procedure was performed. It was found that the intercommunication between the VB application and the ExtendSim model could be somewhat unreliable. Furthermore, the run times of the model were found to be very long when conducting long simulation run lengths. Hence, it was thought that deployment of the application, one of the key requirements, could be compromised by these issues and it was decided to port the model and application to the Python language as a standalone cross-platform program. The following chapter describes this program and how this was carried out.

Semiconductor Fab Model B

This chapter describes the modelling of a full semiconductor fab based on the *Semiconductor Wafer Manufacturing Data Format Specification*, produced by Feigin et al. (1994). It uses the Python scripting language and a third party discrete event simulation (DES) modelling module called SimPy.

6.1 Justification for the use of Python and SimPy

Python is a platform independent, dynamic programming language that applies an object orientated structure (Parkin, 2010). SimPy is a third party open source module for designing DES models in Python. It uses the object orientated nature of the Python programming language and to perform process-orientated DESs, whereby each simulated activity is modelled by a *process*. This makes it a very powerful tool for building a full fab model with minimal scripting. The power of SimPy comes in taking advantage of

Python generator threads which were introduced to Python in version 2.4.

Generators are similar to functions with the key difference that functions return a value and then the instance of the function is destroyed. With generators, the instance of a generator can return values at multiple points during its execution and its state is maintained upon return to the generator. This means that an entity can have a lifecycle controlled by a generator function, that returns control to a central controller when it wants to perform an activity or interact with other activities in the model. Therefore, all of the entities described in Chapter 5, i.e., lots, tools, operators, downtimes and breaks are modelled using Python classes and each instance of these classes has its own lifecycle or programming thread defined by the classes generator.

6.2 Model Input and GUI

The *Semiconductor Wafer Manufacturing Data Format Specification* was ported to a MySQL database, which enforced the semantics of the specification more strongly than using the downloaded text file structure. This helped to realise any errors in the sample datasets and enforce the correct data type (string, float and integer) for any future additional datasets. Figure 6.1 shows a visual representation of the database.

The program interface was built using a third party Python module known as TkInter, a graphical user interface (GUI) for Python. The interface, as shown in Fig. 6.2 allows the user to access any of the datasets stored in the MySQL database.

6.3 Modelling Entities and Processes using SimPy

Each entity in the system has its own class description and lifecycle. The term *lifecycle* is used to describe the events/activities undertaken by the object during its ‘existence’, which is sometimes referred to as its process execution method (PEM). A master/slave relationship is created between the entities in the system, whereby the master finds and

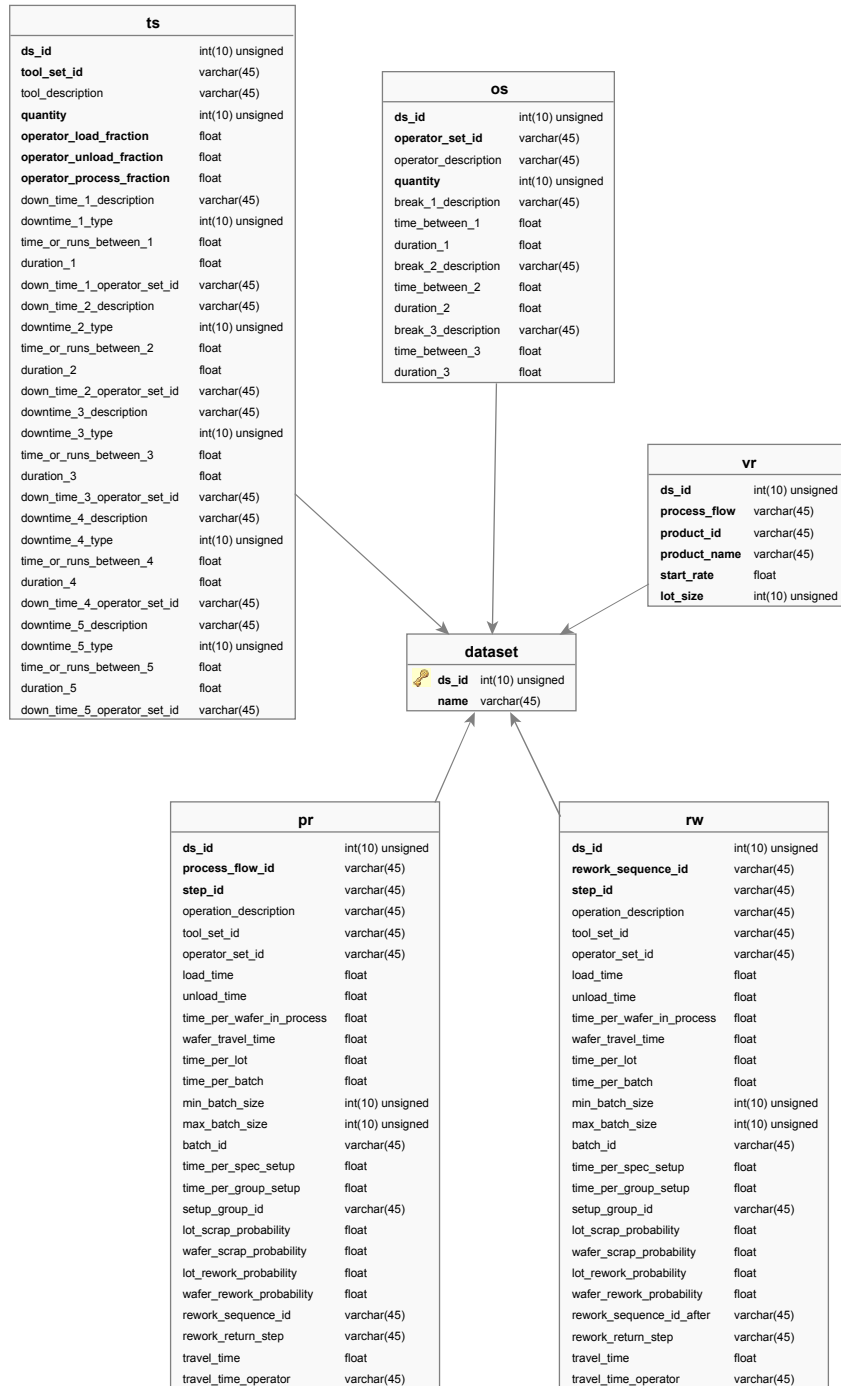


Figure 6.1: Visual interpretation of *Semiconductor Wafer Format Specification* in MySQL database.

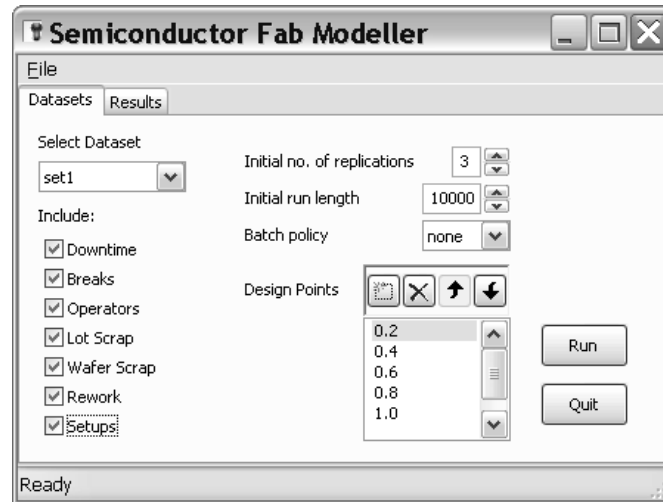


Figure 6.2: GUI for the Python/SimPy application.

extracts the slave from a *stasis* mode for a duration of time to represent some process or activity and then relinquishes the slave back to its stasis mode. The following sections describe briefly the actions and activities of the entities using high level pseudocode.

6.3.1 Lot and operation PEM's

Lots are created using a `lotSource` class and an instance for each product is created. The `lotSource` method executes the following PEM outlined in pseudocode,

```

WHILE the simulation is running
  increment lot counter
  create a lot using the class Lot
  wait for the inter-arrival period

```

which creates an instance of the `Lot` class, whose PEM is given by;

```

FOR each operation in the lot's process route or rework route
  IF it is a batch operation
    WHILE there are insufficient other matching lots to form a batch
      wait until another lot arrives
    WHILE NOT got a tool
      IF an appropriate tool is available
        get the tool from the tool queue
      ELSE
        wait until the tool queue changes
  IF setup is required
    hold for setup period

```

```

FOR loading, processing and unloading activity
  IF activity is required
    IF activity requires operator
      wait for operator
      hold operator for activity fraction
      release operator
      hold for remaining activity fraction
    ELSE
      hold for activity duration
  IF it is a batch operation
    unbatch the individual lots
  send signal to tool to continue
  IF wafer travel is required
    hold for wafer travel duration
  IF full lot scrap test fails
    scrap the full lot
    update scrap counter
    EXIT the PEM
  FOR each wafer in the lot
    IF wafer scrap test fails
      scrap the wafer
      update scrap counter and the lot's quantity
  IF full lot rework test fails
    execute the rework loop for the lot
    EXIT the PEM
  FOR each wafer in the lot
    IF wafer rework test fails
      create a new lot from the lot class with the failed wafers
      execute the rework loop for the newly created rework lot
  IF lot transportation to next step is required
    IF transportation requires operator
      wait for operator
      hold for transportation period
      release operator
    ELSE
      hold for transportation period
  IF current step is the last operation on the lot's process route
    EXIT the PEM
  IF current step is the last operation on the lot's rework route
    rejoin the process route from the last operation
    EXIT the PEM

```

6.3.2 Tool PEM's

Tools are created from the *toolSource* class. An instance is created for each toolset;

```

FOR each tool in the toolset
  create a tool using the Tool class

```

which in turn creates an instance for each tool in the toolset according to the `Tool` class.

The `Tool` lifecycle is given by;

```

WHILE simulation is running
  put the tool into a tool queue (to make it available)
  wait until activated by another entity
  IF activated by a lot
    increment the tools run counter
    IF the tool's run-based downtime has expired
      IF it requires an operator
        wait for an appropriate operator
        hold for the repair period
        release the operator
      ELSE
        hold for the repair period

```

6.3.3 Downtime PEM's

Downtimes are created from the `downtimeSource` class. An instance is created for each tool's downtime mechanism;

```

FOR each downtime mechanism of the tool
  create a downtime instance using the Downtime class

```

The `downtime` PEM is given by;

```

WHILE simulation is running
  hold for mean time before failure period
  remove corresponding tool from tool queue
  IF it requires an operator
    wait for an appropriate operator
    hold for the mean time to repair period
    release the operator
  ELSE
    hold for the mean time to repair period
  send signal to the tool to continue

```

6.3.4 Operator and break PEM's

Operators are created from the `operatorSource` class. An instance of the `operatorSource` class is created for each operator set which uses the `Operator` class to create the required number of operators.

```

FOR each operator in the operator set
  create an operator instance using the Operator class

```

The Operator PEM is given by;

```

WHILE simulation is running
  put the operator into the operator queue
  wait until activated by another entity

```

Break items are created from the `breakSource` class. An instance is created for each operator's break pattern as follows;

```

FOR each break pattern of the operator
  create a break instance using the Break class

```

The Break PEM pseudocode is given by;

```

WHILE simulation is running
  hold for mean time between break sample
  remove corresponding operator from operator queue
  hold for the break duration
  send signal to the operator to continue

```

6.4 Capturing Model Output and Displaying Operating Curves

The model output information is written to a MySQL database. This database is visualised in Fig. 6.3.

An output GUI (Fig. 6.4) can then be used to navigate this database and construct a number of operating curves pertaining to the model runs. The user can then examine the operating curves for individual toolsets, operators or examine the full factory curve.

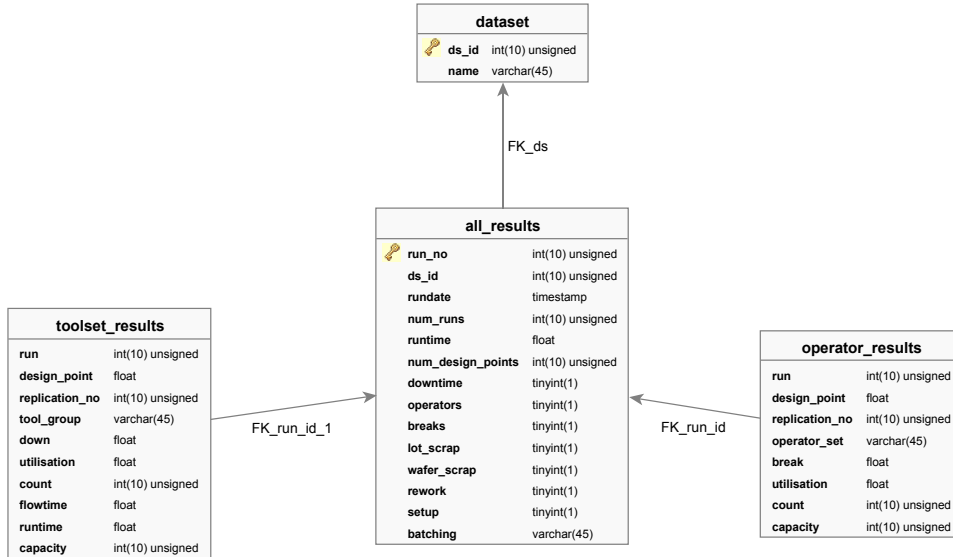


Figure 6.3: Visualisation of output data from simulation model stored in MySQL database.

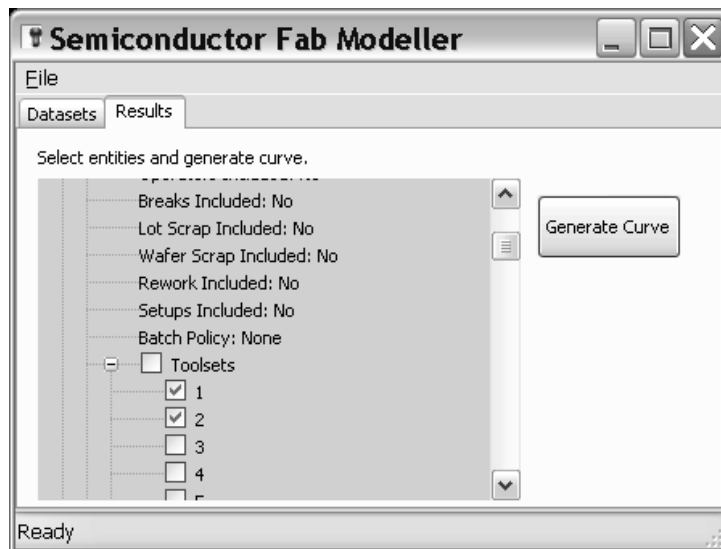


Figure 6.4: Output GUI for SimPy model.

6.5 Analysis of the Minifab Dataset

The minifab dataset is the simplest of all the datasets provided by the specification. Table 6.1 gives a brief description of the dataset according to the configuration file that accompanies it. There are three products; ‘Pa wafer’, ‘Pb wafer’ and ‘Test wafer’ that undergo a single process flow. The process flow consists of six individual operations S1-S6 that pass through five workstations; Ma, Mb, Mc, Md and Me, each with a capacity of 2, 2, 2, 2 and 1 tools respectively.

Table 6.1: Description of Sematech minifab dataset from *MASM Lab Factory Datasets* (1996).

Type of product	wafers
Number of process flows in dataset	1
Number of different products	3
Dataset products make up what % of factory	n/a
Average number of process steps per mask layer	n/a
Are operators modelled?	Yes
Is rework modelled?	No
Number of equipment groups (toolsets)	3
Approximate wafer starts per day	300
Raw process time	Product 1 = 14.4 hrs Product 2 = 14.4 hrs Product 3 = 14.4 hrs

A process step-centric representation (proposed by Ignizio (2009)) is given in Fig. 6.5. All value-added operations (denoted by light circles) require two resources (triangles); a machine resource and operator resource. In between each operation, a non-value-added (dark circle) transport step is required. Toolset Me is required for two operations; S3 and S6, defining a single re-entrant loop for the process flow. The degree of re-entrancy (DoR) is then given as $\frac{6}{5} = 1.2$ for the system.

Table 6.2 gives the raw process time (RPT) for each of the three products according to the simulation, the configuration file and the calculated results. The discrepancy between the calculated results and those given by the configuration file is due to the inclusion of non-value-added transportation steps. However, there is close agreement, a difference of 2.3%, between the simulated results and the calculated results. This small difference was

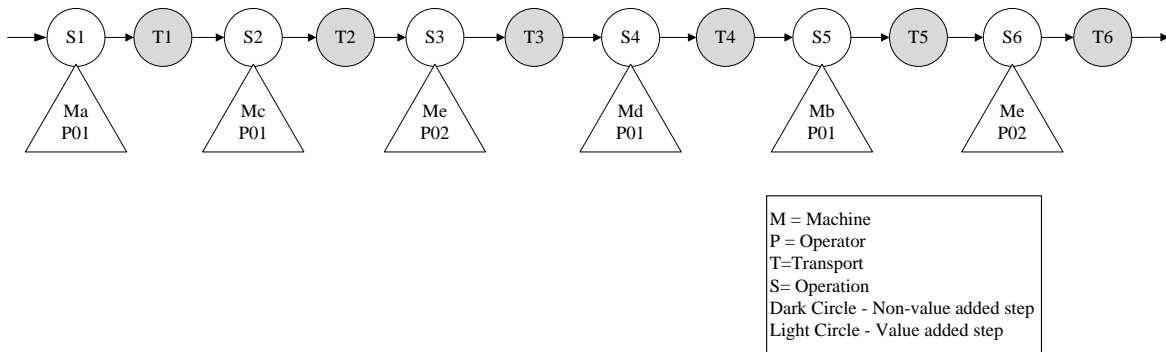


Figure 6.5: Process step-centric representation of minifab dataset.

due to the toolset setup requirements in the simulation model.

Table 6.2: Comparison of reported, calculated and simulated raw process times for minifab dataset.

product	ratio	configuration file (hrs)	calculated (hrs)	simulated (hrs)
Pa wafer	0.606	14.4	15.43	15.76
Pb wafer	0.357	14.4	15.43	15.76
Test wafer	0.036	14.4	15.43	15.76

6.5.1 Operating curve results for minifab dataset without operators or downtime

Table 6.3 lists the cycle time and x-factor results from the SimPy simulation without operators or downtime requirements. Fig. 6.6 plots the resulting operating curve based on lot starts. Due to a number of batching operations the operating curve increases the lower the loading on the system. The minimum cycle time can be found when the system is loaded with a starts rate of approximately 6-7.5 lots per day. However, the minimum cycle time in this range is still almost twice the RPT. Again, this is due to batching requirements within the system.

At very high loading, approximately 8.75 lots per day (equivalent to a bottleneck utilisation of 0.94), the simulation became very unstable and failed to attain steady state even over a very long run length. This is shown by the very sharp increase in the operating

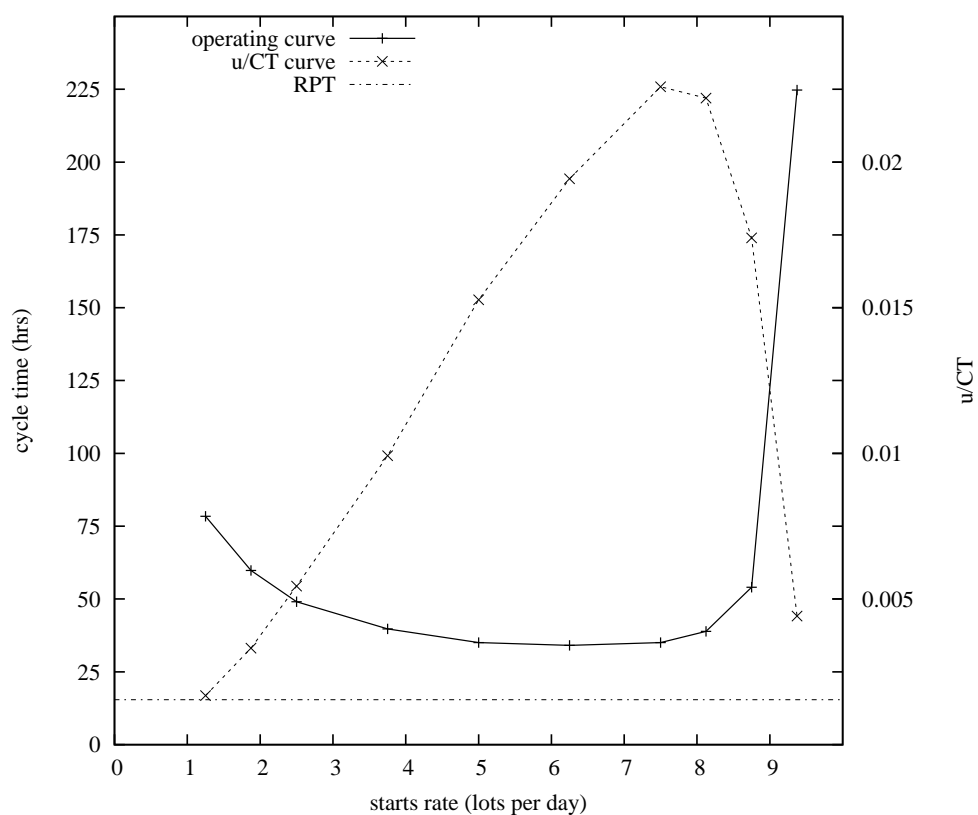


Figure 6.6: Standard operating curve and u/CT curve for minifab dataset without operators or downtime.

Table 6.3: Cycle time and x-factor results for Sematech minifab dataset with no operators or downtime.

starts rate (lots per day)	run length (hrs)	warm-up (hrs)	bottleneck	u_{BN}	CT (hrs)	x-factor
1.250	50000	20000	Me	0.132	78.425	5.082
1.875	50000	10000	Me	0.198	59.808	3.875
2.500	50000	10000	Me	0.267	49.050	3.178
3.750	50000	12000	Me	0.394	39.713	2.573
5.000	50000	7000	Me	0.536	35.069	2.272
6.250	50000	10000	Me	0.663	34.110	2.210
7.499	50000	20000	Me	0.792	35.059	2.272
8.124	50000	20000	Me	0.864	38.912	2.521
8.749 [†]	100000	n/a	Me	0.940	54.039	3.501
9.374 [†]	100000	n/a	Me	0.993	224.753	14.563

[†] The simulation could not achieve steady state.

curve at these loading levels. The u/CT curve verifies that the optimum loading range should be in the approximate region of 6 to 8.5 lots per day. Any lower, efficiency is lost due to the batch forming process, any higher, the bottleneck toolset, which is a re-entrant tool, reaches its maximum capacity and this results in very long queueing times at the toolset.

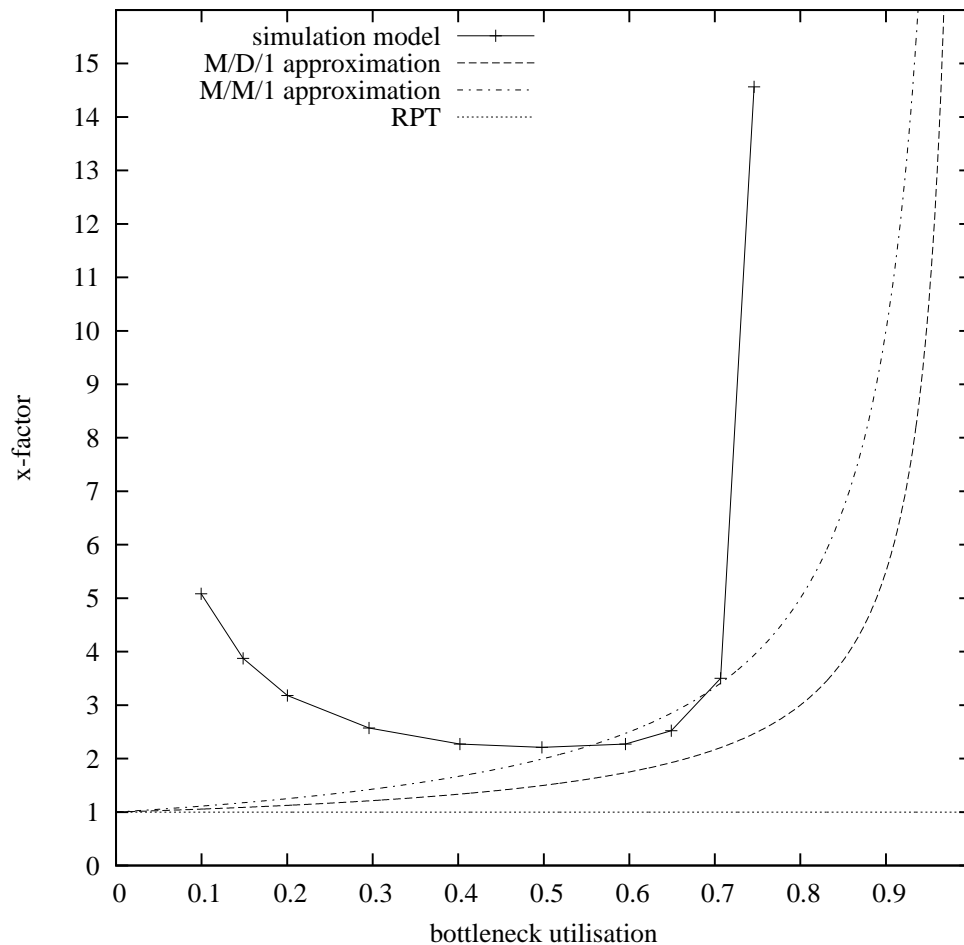


Figure 6.7: Comparison between x-factor results for simulation, M/M/1 and M/D/1 approximation.

Assuming that the bottleneck toolset is representative of the system, it is possible to compare it with an equivalent analytical approximation. Assuming an M/M/1 or M/D/1 queueing system, the cycle time approximations are given by Eq.(2.3), which is compared to the simulation results from Table 6.3 in Fig. 6.7. The plots show, as expected, that neither of the queueing approximations are capable of modelling the ‘wait to batch’ impact on the operating curve. Similarly, both overestimate the operating

curve at higher levels of loading. This shows that it is not possible to place a ‘black box’ queueing model to the fab, similar to the examples shown in the Section 2.4.2, without losing significant accuracy and leaving out some of the dominant fab phenomena such as batching and re-entrancy.

6.5.2 Operating curve results for minifab dataset with operators and downtime

Fig. 6.8 shows the impact on the operating curve with the inclusion of downtime and operators from the simulation results listed in Table 6.4.

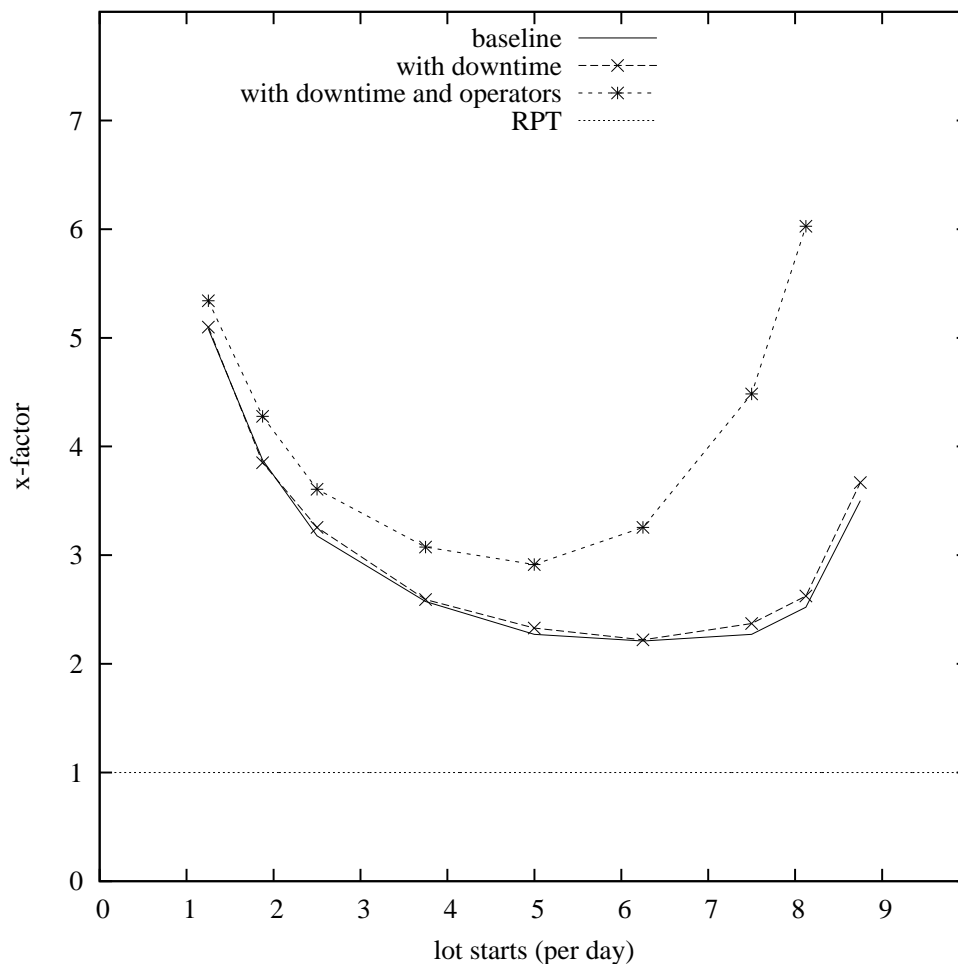


Figure 6.8: Operating curves for minifab dataset with operators and downtime.

All of the toolsets have a downtime mechanism called PM, and toolsets Mc and Md have an additional downtime mechanism called EM. The total availability for each tool

is shown in Table 6.5. The inclusion of downtime has very little impact on the baseline curve, given that the average availability of the tools is 0.945. This is further confirmed by analysing the unavailability coefficient V which has a toolset average of 0.003188. This low value, which is a cycle time multiplier for x-factor approximations, shows that the system is not largely impacted by tool availability.

Table 6.4: Cycle time and x-factor results for Sematech minifab dataset with no operators or downtime.

	starts rate (lots per day)	run length (hrs)	warm-up (hrs)	bottleneck	u_{BN}	CT (hrs)	x-factor
inc. downtime	1.25	50000	20000	Me	0.134	78.69	5.10
	1.87	50000	11000	Me	0.206	59.43	3.85
	2.50	50000	12000	Me	0.270	50.24	3.26
	3.75	50000	10000	Me	0.417	40.00	2.59
	5.00	50000	12000	Me	0.558	35.96	2.33
	6.25	50000	6000	Me	0.695	34.28	2.22
	7.50	50000	16000	Me	0.814	36.61	2.37
	8.12	100000	40000	Me	0.887	40.51	2.62
	8.75	100000	45000	Me	0.948	56.60	3.67
	9.37 [†]	100000	n/a	Me	0.997	169.73	11.00
inc. downtime & operators	1.25	50000	24000	Me	0.189	82.43	5.34
	1.87	50000	21000	Me	0.276	66.02	4.28
	2.50	50000	10000	Me	0.360	55.66	3.61
	3.75	50000	8000	Me	0.552	47.44	3.07
	5.00	50000	12000	Me	0.713	44.98	2.91
	6.25	50000	20000	Me	0.863	50.24	3.26
	7.50 [†]	100000	n/a	Me	0.969	69.19	4.48
	8.12 [†]	100000	n/a	Me	0.992	93.00	6.03

[†] The simulation could not achieve steady state.

However, there is a significant change to the operating curve with the inclusion of operator requirements. There are three operator sets P01, P02 and MT1 consisting each of one operator each. Operators P01 and P02 are required for loading and processing the lots on the tools, but the fraction of time they are required to perform these activities is approximately 4% of the loading and processing time. This means that their average utilisation is quite low at about 0.01. However, the maintenance operator MT1 is required to perform all the maintenance and repair tasks and this has a large impact on the system. Given that the operator set MT1 consists of only one operator, in effect, tools

cannot be repaired at the same time and are often left waiting for the repair operator for long periods of time. This impact increases, the more loaded the tools become and increases the system's x-factor by 3 at high loading volumes. Given that availability does not change over the loading profile, it was found that operator MT01 was utilised approximately 83% of the time. This examination leads to the conclusion that the operator set MT03 may be under-resourced. Given that the two other operators are under-utilised, a recommendation may be to cross-train the loading and processing operators to perform maintenance and repair tasks also.

Table 6.5: Average availability for minifab toolsets.

toolset	no. of downtime mechanisms	A	V
Ma	1	0.950	0.000219
Mb	1	0.950	0.000198
Mc	2	0.932	0.004299
Md	2	0.932	0.003224
Me	1	0.960	0.008000

6.6 Comparison between ExtendSim and SimPy Models

To show that the modelling strategy is independent of the modelling language or application used, the operating curves produced by the ExtendSim model described in Chapter 5 and the SimPy model for the minifab dataset are shown in Table 6.6 and Fig. 6.9.

As can be seen, the cycle time results are in close agreement with an average difference of about 1.54% in the ranges where both simulations achieved steady state. Larger differences were found in the unstable regions of the models, however, the fact that both models were unstable in the same regions verified the similarity of the models.

Applying the paired comparison test, (described by Montgomery (1991)), to the SimPy and ExtendSim operating curves with a high confidence interval of 99%, the results showed that there was no statistically significant difference between the operating

Table 6.6: Comparison of simulation model results for ExtendSim and SimPy models.

starts rate (lots per day)	x-factor		% difference
	ExtendSim	SimPy	
1.250	5.202	5.082	2.322
1.875	3.764	3.875	2.965
2.500	3.160	3.178	0.588
3.750	2.546	2.573	1.055
5.000	2.300	2.272	1.216
6.250	2.178	2.210	1.463
7.499	2.252	2.272	0.870
8.124	2.570	2.521	1.884
8.749	2.937	3.501	19.231
9.374 [†]	32.043	14.563	54.552

[†] The simulation could not achieve steady state.

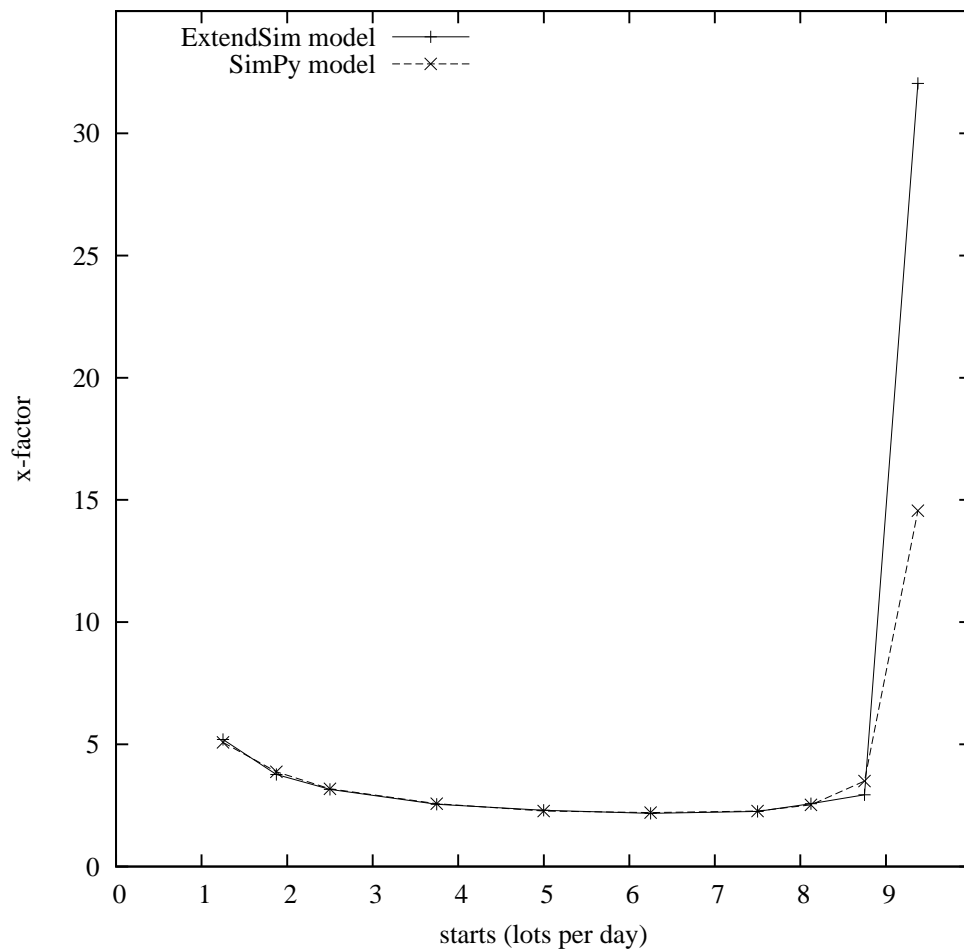


Figure 6.9: Comparison of simulated operating curves for ExtendSim and SimPy models using the minifab dataset.

curves, despite the fact that a visual difference can be found at very high loading levels.

This verified that the modelling strategy described here and in Chapter 5 could be deployed using a number of different modelling applications. In terms of performance though, the SimPy model outperformed the ExtendSim model in terms of execution speed. The speed performance was generally found to be in the range of 5:1 in favour of the SimPy model, particularly for the more complex datasets. A further disadvantage of the ExtendSim model is that attributes must be defined by real or integer values only, strings are not allowed. This meant that any string values needed to be converted to a unique value prior to model execution. This was not an issue in the SimPy models which handled strings and string operations efficiently.

6.7 Model Verification & Validation

Some of the verification techniques, summarised by Whitner and Balci (1989) and listed in Appendix G.1, were applied and the results are discussed in Table 6.7.

Table 6.7: Techniques used to verify the SimPy model.

Type	Technique	Results and Comments
Informal	Code inspection	The source code was reviewed and inspected during model build and after the final version. Much of the difficulty with debugging the source code arose as a result of SimPy's complex interaction between different entities.
Static	Syntax analysis	Modern software compilers ensured that the model syntax was complete and verified. SimPy for Python was debugged using PyDev debugging tool which is a plugin for the open source Eclipse integrated development environment (IDE). Debugging was complicated by the multiple threads of the SimPy generators, however, the IDE allows monitoring of unique variables and objects with conditional breakpoints, which facilitated the debugging process.

continued on next page

continued from last page

	Structural analysis	Structural analysis was performed according to the best practices of coding. The Python <code>logging</code> library allowed customised comments or logs to be written to a text file which was then used to monitor the movement and activities of entities during model execution. Furthermore, SimPy has a trace feature which prints to the IDE console, all of the discrete events that occur during runtime. Each version of the model was checked using this tracing feature during alpha testing.
Dynamic	Top-down testing Bottom-up testing Black-box testing	Top-down, Bottom-up testing was applied using a black-box style that checked each class and function to ensure each portion was consistent and did not produce erroneous outputs.
	Stress testing	Lot arrival rates were gradually increased until the model became overloaded and unable to reach steady state. Mean time before failure (MTBF) and mean time to repair (MTTR) values were increased gradually causing very small tool availability. This resulted in very long lot queueing times in the model as expected. Other parameters such as scrap and rework probabilities were increased to verify that flow became increasingly restricted in the model. This showed that the initial interpretation of rework re-merging with parent lots was severely restricting lot flow in the datasets. Hence, it was decided that rework should not merge back to its parent lot upon completion of its rework operations.
	Debugging	Debugging was an ongoing process during the model build and each version of the model and application was subjected to rigorous debugging as outlined above.
	Execution tracing	Similar to the ExtendSim models, data flow analysis and model tracing was performed for the simulation model by introducing only one lot into the system and monitoring its attributes as they changed during runtime, proving that the lot experienced no queueing and minimal processing time, the model data flow was correct. A zero batching policy was employed during this process.
	Execution monitoring	As discussed above, trace animation capabilities in SimPy, (i.e., use of the SimPy <code>SimulationTrace</code> module as opposed to the <code>Simulation</code> module) were used to validate the object flow and activities during runtime.
	Regression testing	Regression testing (repeating the above procedures) was performed during alpha and beta testing and after each new model version.

Validating the model was performed by comparing the ExtendSim and SimPy model cycle time and x-factor results for the minifab dataset. As can be seen from Section 6.6, the results are in close agreement. Other validation measures are included in Table 6.8. An explanation of each test can be found in Appendix G.2.

Table 6.8: Techniques used to validate the SimPy model.

Technique	Results and Comments
Degenerate tests	Degenerate tests were performed by overloading the model by gradually increasing the arrival rate of test lots into the model and forcing utilisation of the toolsets and operators to capacity. This forced the model into an unstable state where the cycle time increased rapidly and the model was unable to achieve steady state. This process was used to examine the boundaries of the model inputs.
Event validity	Event validity was ensured by monitoring the routing/events of entities in the model and matching against the dataset information. Each event was assigned a check list of completed operations during execution that were examined afterwards for consistency.
Internal validity	Very few replications (typically about 3) according to the calculations described in Section 3.2 were required, meaning that the output results variability of the performance metric (cycle time) was very low.

6.8 Summary

This chapter described a Python and SimPy implementation of an automated, flexible and reusable modelling application for generating fast and reliable operating curves for a full semiconductor fab. The modelling strategy involved analysing the system in an entity-centric manner, whereby, the dominant components of the system are modelled as objects. The interaction of these entities models the operations and activities that the fab undergoes. This modelling strategy was shown to be far more effective than a traditional modelling strategy and allowed a framework to be developed that could auto-generate the simulation models. Theoretically then, any fab that could be captured and described in the data specification, could then be modelled and its operating curve could be produced without any need for rebuilding the model. The concept of this method was proved in two independent trials, one using an ExtendSim model embedded in a Visual Basic (VB) application, the other in a SimPy model controlled by a Python application.

The Python application was used to analyse the minifab dataset, and the resulting operating curves showed that the system had an efficient operating region below which ‘waiting to batch’ times significantly reduced the efficiency of the system, and above which, the bottleneck toolset became overloaded and long queueing times resulted. It was also recommended that cross-training or up-skilling of operators would increase the

efficiency of the system and reduce the utilisation of the maintenance operator set.

Comparisons between the simulated operating curves and analytical approximations showed that simple black-box style queuing models could not be used to capture some of the complexities of the fab, (e.g., re-entrancy and operators), and that the simulation strategy employed here offered the a better method for success.

Comparisons between the ExtendSim and SimPy implementations of the modelling strategy showed negligible difference between the model results, which was verified both visually and with a statistical paired comparison test. This showed that the conceptual modelling strategy used is independent of the modelling and programming tools used, and is successful at generating consistent and repeatable operating curves.

Discussion

7.1 Overview

This thesis showed how operating curves can serve as a good holistic metric for complex semiconductor wafer fabs, and can aid engineers and management to make better decisions regarding capacity planning and allocation of resources. The strength of operating curves lies in their fundamental relationship to the economic driving factors that the semiconductor industry is faced with.

However, the generation of operating curves is a non-trivial matter. Many of the previous attempts to generate these curves involved using queueing models with broad assumptions about the system under analysis. These curves were then limited by these assumptions and the result is that they were only capable of benchmarking and comparing previously generated curves using the same methodology.

It was shown that simple analytical approximations are incapable of describing multi-

faceted systems where the capacity of the system is dictated by a number of factors, such as operator availability, re-entrancy and more complex dynamic scheduling. In fact, much of the published literature on queueing approximations applied to a semiconductor fab, seem to expand simple operating curves to account for just one more additional complexity, such as downtime (Section 2.4.1) or the ‘idle with work in process (WIP)’ scenario (Section 2.4.1). Including a multitude of additional complexities to the simple approximation appears to be a difficult task and there is no literature as of writing that attempts to capture the many complexities of a semiconductor toolset in a holistic analytical descriptor. The very fact that most of these approximations rely on fixed deterministic values for toolset capacity fails to reflect the dynamic environment of a semiconductor toolset where the perceived capacity of the toolset is often in flux and dictated by factors such as the setup conditions of the tools and the complex run rules that the tools have with the lots or wafers.

Furthermore, the highly re-entrant nature of fabs means that lot departure patterns from workstations can have a significant impact to its lot arrival pattern. This is an issue that has yet to be successfully addressed by queueing theory approximations and which almost no literature exists according to Shanthikumar et al. (2007). Many of these issues arise as a result of the fundamental assumptions upon which queueing theory is based.

With these issues in mind (and supported by the literature) it was decided that simulation modelling, was a more appropriate method for capturing the complex behaviour of a semiconductor fab. The strength of simulation modelling means that any level of system complexity can be recreated or modelled provided there is sufficient time and resources to do so. However, as outlined in Section 2.5, simulation modelling is not without its own problems. Mainly, the lead time and effort required to create a simulation model, and subsequently maintain it, are the biggest detractors to its use. Most of the time taken to complete a simulation study is often consumed by data collecting and collating.

This made the case for creating generic models that were driven by a consistent data specification or ontology. Key to this approach was to create applications that could

generate simulation models depending on the data specification. The first attempt, the Flexible Toolset Modelling (FTM) application, was used to generate operating curves for standalone toolsets. Despite successfully achieving this objective, there were issues regarding analysing a toolset in isolation from its upstream and downstream conditions.

Therefore, it was decided to generate operating curves for a full fab using a flexible model-generating application that could capture the most relevant parts of the system. The *Semiconductor Wafer Manufacturing Data Format Specification* was used as an information model to implement a flexible generic model that negated any interaction with the model or editing of the model structure.

This modelling framework was implemented in both a Visual Basic (VB) program that used ExtendSim as its backbone simulator, and a standalone Python application which used the SimPy modelling package. Both these applications generated fast and accurate operating curves for dataset 1 and the minifab dataset. Comparisons between the two models showed close agreement of their resultant operating curves, which helped to verify the modelling strategy employed.

7.2 Optimum Location of Simulated Design Points on Operating Curves

Location of design points on the operating curve and selection of loading levels to produce the most representative operating curve with the minimal amount of simulation effort is a topic of little interest in the associated literature with just a few publications choosing to focus on the variability output from the design points. Perhaps this is due to increased computing resources, meaning that simulation effort and processing power is less of a premium to the analyst. However, any time-savings that can be made should be examined and exploited if one is attempting to automate a full simulation project. The tactic used in this thesis (in Section 3.1.2) involved applying a simple black-box style queueing model over the system under analysis to get a preview of the likely shape of an operating curve

of such a system. It was then possible to estimate the likely ‘important’ sections of the operating curve that an analyst would be most interested in.

The u/CT curve proved an effective tool for this problem. The ratio of the two performance indicators was shown to approximate better the area where the fastest change of the curve occurred. Allocating more simulation effort to design points in this area was shown to capture the inflection region of the operating curve and place minimal simulation effort on the low utilisation regions where the horizontal asymptotic nature of the operating curve is predominant. Similarly, little simulation effort is requested in the higher utilisation area of the curve where the vertical asymptote dominates. It appears as a contradiction to allocate lesser simulation effort in this later area where typically the bulk of simulation effort is required due to the instability of the simulation model, however, on reflection it makes most sense. This area is quickly approaching the upper bounds of system capacity and if the model is a good representative of the real system it is unlikely that the real system will be loaded to this level for long periods of time. Therefore, it is unlikely that the analyst will be as interested in this area of the operating curve, particularly given the fact that a disproportionate amount of simulation effort is required to produce any sound simulation estimates. This was evident in most of the operating curves based on the Sematech datasets. It appeared that in most there was a well-defined threshold of loading, above which the simulation models became unstable very quickly and it appeared that regardless of the amount of simulation effort exhausted (either run length or number of replications), the simulation models could not achieve steady state, in this region.

7.3 Operating Points, Curves and Surfaces

The assumption throughout this thesis has been that fabs, at any given time, exist as a point on the operating ‘spectrum’. Changing the loading of the fab then causes this point to shift in the x-y plane and the line defined by the movement of this point

from zero loading to full loading defines the operating curve. Alternatively, changing the configuration of the system causes a shift in the curve and a new line is defined. Visualising this, is then usually done in a two-dimensional plane using two different curves to describe the two alternate configurations. However, it might be a broad assumption to assume that changes in loading level will not cause a shift in the curve. It is likely that changing the loading level will indirectly cause a shift (perhaps a minor one) in the operating curve. For example, increasing the loading level at a toolset by decreasing the lot arrival rate (per unit time) may promote the use of an alternative dispatching rule like setup minimisation, thereby changing the system configuration. Hence it might be more useful to visualise the operating spectrum as a 3-dimensional surface as opposed to a series of separate and distinct operating curves. The movement of the operating point would then be a 3-dimensional traversing of a surface as opposed to leaping between different operating curves. This would also help engineers to understand the multiple factors that define an operating curve.

The operating surfaces for a queueing system shown in Section 3.5 use variability as the third dimension, but this could easily be substituted with other operating curve factors such as the parallel capacity m or WIP profile. For example, assuming that a toolset consisting of ten tools processes a number of products with an identical service pattern. Then the number of tools that the incoming lots can select from is equal to the toolset capacity. However, assuming that a product (product A) is dedicated to one tool that is qualified to run it, then, as the percentage of product A increases in the WIP profile, the less efficient the system becomes and the result can be visualised using an operating surface of the system, as in Fig. 7.1. Such a visualisation would help engineers to understand better the multiple proponents of operating curves and make more informed decisions at the operational level based on an impending WIP profile or product mix.

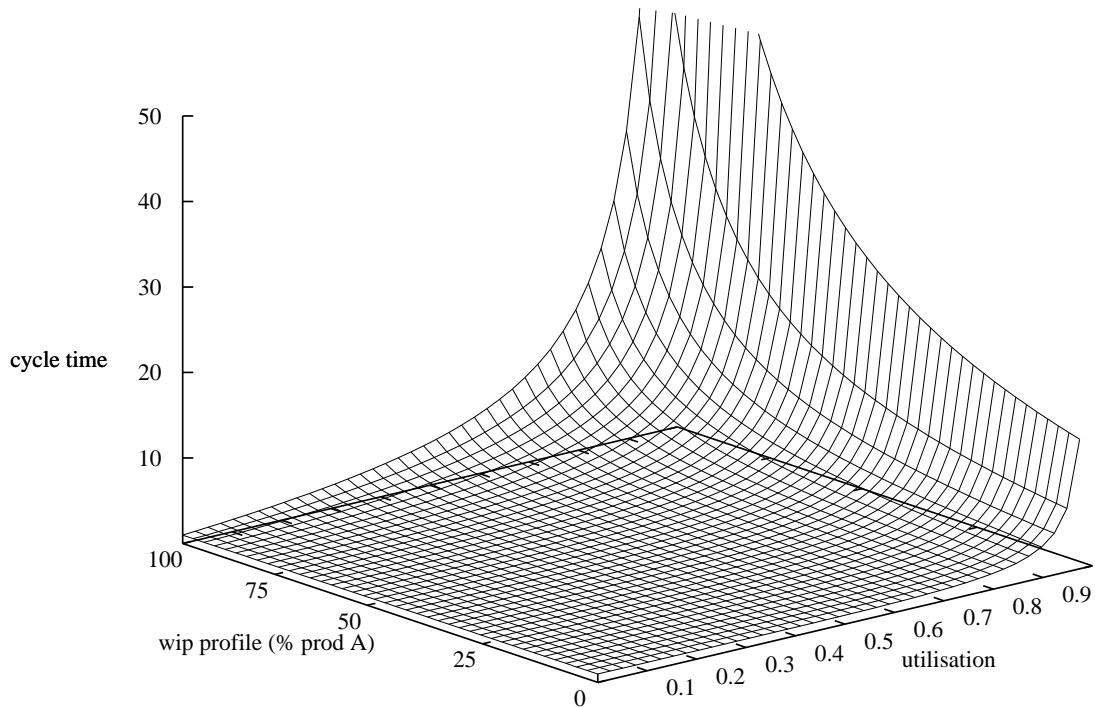


Figure 7.1: Operating surface for G/G/10 queue with lot dedication restriction.

7.4 Reflections on the FTM application

One of the most difficult aspects when implementing the FTM application involved modelling the offline times of the tools. Of the case study examined, identification and classification of unscheduled and scheduled downtime is unclear. The timing of a preventative maintenance (PM) event is dependent on the status of the tool, the quality output from the tool, and the level of incoming WIP. It was found that the flexibility of PMs, meant that there was a high level of overlapping between scheduled and unscheduled outages. For example, often was the case where a tool that was interrupted during processing, which should technically be referred to as an unscheduled downtime, would be recorded as having entered a PM state, and repair of the tool could be achieved by expediting a planned maintenance task. This caused some confusion over the nature of the offline event and required some heuristic algorithms in the application that could correctly identify the likely sequence of events and categorise the type of offline event correctly.

Similar difficulties occurred when interpreting what happened when a tool came back online and went down again soon afterwards. In such a situation it was impossible to

identify whether the follow-up offline event was part of the previous one. Again, some heuristic algorithms were required to account for this scenario by assuming an upper threshold mean time before failure (MTBF) interval, under which, successive down events were considered to be part of the same failure.

Problems such as these showed that despite the existence of logging protocols at the test facility, the states of the tool could be interpreted differently by different operators. Many of these issues only came to light during the programming and debugging of the data collection scripts, and the time taken to address the issues far outweighed the time taken to build the actual generic model itself. It is likely that had the FTM application been subsequently applied to a different toolset, a similar phase of debugging might have been required for other unforeseen issues.

As a proof of concept the FTM application was shown to be a useful tool for generating the operating curves for the test case. The shape of the curves showed that the tools under investigation were efficient and there was little variation in actual process times. This led to the conclusion that much of the variability in the system was due to complex despatching rules and lost efficiencies in lot transportation to and from the tool. This concern helped identify the difficulty of examining a toolset in isolation from its upstream and downstream conditions. Which in turn, promoted the case for analysing a full system and applying the generic modelling methodology on a fab-wide scale.

7.5 Craft-based versus Generic Modelling

Despite discrete event simulation (DES) modelling being a scientific procedure it is still subjected to the interpretation of the system by the modeller. In many cases it would be likely that two independent simulationists would create two very different models of the same system. This emphasises the fact that simulation modelling is very much an art, something which is evident in single-use, one-off or craft-based models. Craft-based models can be very accurate, encompass a great level of system detail and capture a

generally uncommon theme or phenomena that the system is subjected to. However, there is a lost opportunity when applying it to semiconductor manufacturing. The components of the system lend themselves well to be modularised, given that they have many common elements and structures. This promotes the case for using generic modelling with a set of variable inputs and structures.

However, creation and deployment of the models in this thesis have led to the conclusion that there is perhaps a correct way and an incorrect way of doing this. The FTM application was limited in its reusability because its initial design was far too targeted towards the unique circumstances of the testbed toolset. Perhaps the correct way is to define the conceptual model and adapt the real data to fit it. This is where data specifications, information models and descriptive ontologies can deliver significant time-savings. Basing the modelling strategy around the *Semiconductor Wafer Manufacturing Data Format Specification* meant that the system definition was more robust, less likely to be misinterpreted and easier to create the programs and scripts that generated the simulation models.

It is likely that the future of successful large scale deployment of reusable generic models should be based on this premise. The requirements in understanding large system complexity and ultimately conceptualising the system place a heavy workload on the modeller, and often it is an environment that the modeller is unfamiliar with or has just been introduced to. The modeller then, in a sense, becomes a translator between the real system and the conceptual model that is contained within the simulation program code. A common ground between the modeller and system owners could perhaps be found in a common system descriptive framework such as an ontology or a data specification. The system owners could be tasked with the collection of system information and be responsible for populating the information model. The modeller could then focus more on creating the scripts and programs necessary to generate the models based on the rules and constructs of the ontology, rather than acquiring a deep understanding of the real system.

Employing this tactic could also have huge cost and time savings when conducting a modelling project. An information model constructed by those most knowledgeable about the system may be superior to one populated through the possibly narrower view of the modeller. Wikipedia is a good example of this phenomena, whereby anyone can post any information, which is ultimately judged by the public. This effectively roots out irrelevant or inaccurate content quickly, particularly if the subject matter is one of interest to many people. Such a scheme could be applied to collection of information for simulation models, whereby, tacit knowledge could be fed into an ontology by factory floor personnel that understand the system best. This would make deployment of simulation models and generation of system metrics such as the operating curve a rapid and efficient process.

7.6 Industrial Implications

The following section discusses the industrial implications of this work with respect to the four grand challenges that face simulation modelling in the manufacturing sector according to Fowler and Rose (2004). The grand challenges are listed as follows;

1. An order-of-magnitude reduction in problem-solving cycles,
2. Development of real-time simulation-based problem-solving capability,
3. True plug-and-play interoperability of simulations and supporting software within a specific application domain,
4. Greater acceptance of modelling and simulation within industry.

The first of these challenges refers to reducing the time taken to “design, collect information/data, build, execute, and analyse simulation models to support manufacturing decision making”. Fowler and Rose state that, while there is opportunity and scope for increasing the efficiency of all stages of a simulation project, most gain is to be had in the area of data collection and synthesis for the model. The work in this thesis shows how it is possible to create a framework for simulation modelling by extracting the most

common and repetitive structures of a complex system and reduce the information into an ontology or information model that can be used to generate automated simulation models.

In reference to supporting manufacturing decision making, it is important to base decisions on a holistic view of the system rather than an incomplete picture. For these purposes, the operating curve has been shown to be a key and concise snapshot of the status and efficiency of a factory. Therefore, generating the curve in a timely fashion could be a valuable aid.

Fowler and Rose go on to state that “conventional simulation software packages used for modelling manufacturing systems take a job-driven world-view. In this approach, manufacturing jobs are the active system entities while system resources, such as machines, are passive. The simulation model is created by describing how jobs move through their processing steps, seizing available resources whenever they are needed. A separate record for every job in the system is created and maintained for tracking wafers or lots through the factory. A lot of execution time can be consumed when sorting lists of these jobs in a given queue or in searching the queue for a given job. Therefore, the speed and space complexity of these simulations must be at least on the order of some polynomial of the number of jobs in the factory”. It appears from the work in this thesis that this job-driven modelling strategy is far more difficult to automate and far less flexible than the reusable modelling strategy used in Chapters 4-6. This type of strategy made it possible to interpret any type of semiconductor system that could be realistically captured in the *Semiconductor Wafer Format Specification*.

Another area that Fowler and Rose believe has opportunity for improvement, is in the time taken to perform the experiments. This is tackled in Chapter 3, where a complete framework for simulation models is automated into a single program that can control simulation models and remove much of the decision-making and labour required to design the simulation experiments. The models can then be embedded in this framework resulting in a significant reduction in the overall lead time of a simulation project.

The second of the grand challenges ‘real-time simulation-based problem-solving capability’, is only possible once the first challenge has been overcome as single-use one-off models are incapable of repeatedly delivering real-time decision-support. None of the models in this thesis are capable of delivering real-time solutions, however, refining the initial ExtendSim/VB model in Chapter 5 and implementing the framework in a Python/SimPy application in Chapter 6 showed a reduction in time-to-solution by a magnitude of about 5:1. While not offering real-time solutions, this meant that for the more complex datasets, it was possible to generate operating curves in about 1 hour (without performing multiple replications), which is a good starting point. To get to real time, one might need to consider current actual WIP profiles of the real system, and pre-populate the model with this information, thereby removing the warm-up period from the model execution. Such an analysis was not performed here as the *Semiconductor Wafer Format Specification* does not include such mechanisms, however, it would not be that much more difficult to implement this in these models given their modular structure and flexibility.

The last grand challenge refers to increasing the awareness, acceptance and credibility of simulation modelling in the manufacturing sector. Management are often not interested in the inner workings or the theory of simulation, but more interested in how it can help them to make better decisions. Fowler and Rose point out that simulation should not be miss-sold to the industry as being a device to solve all manufacturing problems, but should be advertised as an important tool for management to exploit. Realistically, it is only when the other first three grand challenges are overcome, will one start to see broader acceptance of simulation modelling. It is the simulation results that matter to management, not the mechanisms by which they work. In light of this, it is hoped that the output operating curves from the models introduced in this thesis will be of use to management, in that they are based on sound principles for generating operating curves that can be integrated into a standalone package with a fast time to solution.

Conclusion

The main conclusions and contributions of this thesis are summarised as follows:

- A framework was presented for generating operational characteristic curves for semiconductor fabrication facilities using automatically generated flexible models that avail of simple queueing models to estimate the parameters for the simulation experiments.
- Using the test cases, it was shown that the proposed framework better estimated the underlying operating curve in comparison with queueing theory based models, which failed to identify the high x-factor in the low loading regions (due to batching) and overestimated the efficiency of the fab in the high loading regions.
- A novel method of using the u/CT curve was shown to identify the most pertinent location of design points for defining the experimental parameters of the simulation models.
- The operating curves produced for both versions of the full fab model were shown to be consistent with one another, which verified the flexible modelling strategy used.

8.1 Technical Contributions

In order to facilitate the main contributions and findings, a number of technical objectives were met;

- A library of functions and subroutines were written in Visual Basic (VB) for communicating with and controlling ExtendSim simulation models as a background process.
- A function for estimating the Johnson distribution parameters (based on an algorithm by Slifker and Shapiro (1980)) of a dataset was programmed in both Python and VB.
- Operating surfaces were introduced, which show the relationship between the key factors of an operating curve.
- A program for designing discrete event simulation (DES) model experiments (based on the work of Johnson, Feng, Ankenman and Nelson (2004)) was created which:
 - Allocates experimental design points to minimise simulation effort,
 - Estimates the run length,
 - Determines the required number of replications,
 - Identifies the initial bias for deletion.
- Creation of custom Lot Generator, Tool Generator, Operator Generator and Pairing modular blocks were created for ExtendSim to facilitate more flexible models using an entity-centric modelling strategy.

8.2 Recommendations for Future Work

An ontology such as the *Semiconductor Wafer Manufacturing Data Format Specification* was shown to be capable of capturing and describing a complex system such as a semiconductor wafer manufacturer. It shows that by identifying key components and entities that reside within the system, a pattern can be found which can then be classified in an information model. However, some deficiencies of this information model were identified.

Further development of the wafer specification could include more detailed time recordings. Currently, the specification only offers average times. This could be expanded to include distribution information. Other additions might include the complex run rules and despatching rules that are common in a semiconductor fab and which make it such a difficult environment to describe.

More recently, the Core Manufacturing Simulation Data (CMSD) specification has produced a more complete ontology for designing DES models for a generic manufacturing system. However, according to Ehm et al. (2009), one of the main detractors from using the CMSD is an interface to create the information model (which is stored in an extensible markup language (XML) file and a unified modelling language (UML) interpretation). Currently, work is being undertaken by the author of this thesis to create such a user interface that would allow factory engineers, operators and technicians to build a full database profile of their factory. Once completed, the next phase will involve creation of a program to translate the CMSD information model to a simulation model in Python/SimPy, based on the methods described in this thesis. This project is currently being undertaken as part of the *BreakCycle* Maintainable Modelling Strand for the Irish Centre for Manufacturing Research.

References

- Abadir, M. 2007. Meeting the evolving challenges of the semiconductor industry, *International Conference on Design Technology of Integrated Systems in Nanoscale Era (DTIS)*.
- Acklam, P. J. 2003. *Peter's Home Page* [Online]. Available from: <http://tinyurl.com/6s6l3ww> [Accessed 10 September 2009].
- Ahmad, M. M. and Dhafr, N. 2002. Establishing and improving manufacturing performance measures, *Robotics and Computer-Integrated Manufacturing* 18(3-4), pp. 171–176.
- Akhavan-Tabatabaei, R., Ding, S. and Shanthikumar, J. 2009. A method for cycle time estimation of semiconductor manufacturing toolsets with correlations, *Proceedings of the 2009 Winter Simulation Conference*, pp. 1719–1729.
- Alexopoulos, C. 2006. A comprehensive review of methods for simulation output analysis, *Proceedings of the 2006 Winter Simulation Conference*, pp. 168–178.
- Altiock, T. 1997. *Performance Analysis of Manufacturing Systems*, Springer, New York.
- Atherton, L. and Atherton, R. 1995. *Wafer Fabrication: Factory Performance and Analysis*, The Springer International Series in Engineering and Computer Science, Springer.
- Aurand, S. and Miller, P. 1997. The operating curve: a method to measure and benchmark manufacturing line productivity, *Proceedings of the 1997 IEEE/SEMI Advanced Semiconductor Manufacturing Conference*, pp. 391–397.

-
- Backhouse, R. C. 1986. *Program Construction and Verification*, Prentice-Hall.
- Bai, P. 2009. Advancing Moore's law: Challenges and opportunities, *IEEE 8th International Conference on ASIC 2009*, pp. 7–8.
- Banks, J. 1999. What does industry need from simulation vendors in Y2K and after? A panel discussion, *Proceedings of the 1999 Winter Simulation Conference*, Vol. 2, pp. 1501–1508.
- Banks, J., Carson, J. S. and Nelson, B. L. 2004. *Discrete-Event System Simulation*, 4th edn, Prentice Hall.
- Banks, J. and Gibson, R. 1997a. 10 Rules for determining when simulation is not appropriate, *IIE Solutions* 29(9), pp. 30–32.
- Banks, J. and Gibson, R. 1997b. Simulation modeling: Some programming required, *IIE Solutions* 29(2), pp. 26–31.
- Banks, J. and Gibson, R. R. 1996. Getting started in simulation modeling, *IIE Solutions* 28(11), pp. 34–39.
- Banks, J. and Norman, V. B. 1995. Justifying simulation in today's manufacturing environment, *IIE Solutions* 27(11), pp. 16–19.
- Barendregt, H. P. 1984. *The Lambda Calculus - Its Syntax and Semantics*, Vol. 103 of *Studies in Logic and the Foundations of Mathematics*, North-Holland.
- Bergmann, S. and Strassburger, S. 2010. Challenges for the automatic generation of simulation models for production systems, *Proceedings of the 2010 Summer Simulation Multiconference*, Society for Computer Simulation International, pp. 545–549.
- Bhavnagarwala, A., Borkar, S., Sakurai, T. and Narendra, S. 2010. EP2: The semiconductor industry in 2025, *IEEE International Solid-State Circuits Conference Digest of Technical Papers (ISSCC2010)*, pp. 534–535.
- Boning, D., McIlrath, M., Penfield, P., J. and Sachs, E. 1992. A general semiconductor press modeling framework, *IEEE Transactions on Semiconductor Manufacturing* 5(4), pp. 266–280.
- Butler, K. and Matthews, J. 2001. How differentiating between utilization of effective availability and utilization of effective capacity leads to a better understanding of performance metrics, *Proceedings of the 2001 IEEE/SEMI Advanced Semiconductor Manufacturing Conference*, pp. 21–24.

-
- Carson, J. 2005. Introduction to modeling and simulation, *Proceedings of the 2005 Winter Simulation Conference*, pp. 7–13.
- Carson, J. S. 2002. Verification validation: Model verification and validation, *Proceedings of the 2002 Winter Simulation Conference*, pp. 52–58.
- Christensen, C., King, S., Verlinden, M. and Yang, W. 2008. The new economics of semiconductor manufacturing, *IEEE Spectrum* 45(5), pp. 24–29.
- Chwif, L., Barretto, M. and Paul, R. 2000. On simulation model complexity, *Proceedings of the 2000 Winter Simulation Conference*, pp. 449–455.
- Condrón, F. 2010. *Assessment of approaches for estimating the warm-up period in discrete event simulation models*, Master’s thesis, Dublin City University.
- Connors, D., Feigin, G. and Yao, D. 1996. A queueing network model for semiconductor manufacturing, *IEEE Transactions on Semiconductor Manufacturing* 9(3), pp. 412–427.
- de Ron, A. and Rooda, J. 2005. Fab performance, *IEEE Transactions on Semiconductor Manufacturing* 18(3), pp. 399–405.
- DeBrotta, D., Roberts, S., Dittus, R., Wilson, J., Swain, J. and Venkatraman, S. 1989. Modeling input processes with Johnson distributions, *Proceedings of the 1989 Winter Simulation Conference*, pp. 308–318.
- Delp, D. 2004. A new x-factor contribution measure for identifying machine level capacity constraints and variability, *Proceedings of the 2004 IEEE/SEMI Advanced Semiconductor Manufacturing Conference*, pp. 334–338.
- Delp, D., Si, J. and Fowler, J. 2006. The development of the complete x-factor contribution measurement for improving cycle time and cycle time variability, *IEEE Transactions on Semiconductor Manufacturing* 19(3), pp. 352–362.
- Delp, D., Si, J., Hwang, Y. and Pei, B. 2003. A dynamic system regulation measure for increasing effective capacity: the x-factor theory, *Proceedings of the 2003 IEEE/SEMI Advanced Semiconductor Manufacturing Conference and Workshop*, pp. 81–88.
- Delp, D., Si, J., Hwang, Y., Pei, B. and Fowler, J. 2005. Availability-adjusted x-factor, *International Journal of Production Research* 43(18), pp. 3933–3953.
- Dubash, M. 2005. *Moore’s Law is dead, says Gordon Moore* [Online]. Available from: <http://news.techworld.com/operating-systems/3477/moores-law-is-dead-says-gordon-moore/> [Accessed 19 February 2010].

-
- Ehm, H., McGinnis, L. and Rose, O. 2009. Are simulation standards in our future?, *Proceedings of the 2009 Winter Simulation Conference*, pp. 1695–1702.
- El-Kilany, K. 2003. *Reusable modelling and simulation of flexible manufacturing for next generation semiconductor manufacturing facilities*, PhD thesis, Dublin City University.
- ExtendSim 2009. *ExtendSim 8 Developer Reference*, Imagine That.
- Fayed, A. and Dunnigan, B. 2007. Characterizing the operating curve: How can semiconductor fabs grade themselves?, *International Symposium on Semiconductor Manufacturing*, pp. 1–4.
- Feigin, G., Fowler, J., Robinson, J. and Leachman, R. 1994. *Semiconductor Wafer Manufacturing Data Format Specification*, Modeling and Analysis for Semiconductor Manufacturing Laboratory (MASMLab). Available from: www.eas.asu.edu/~masmlab/ [Accessed 2 May 2006].
- Fishman, G. S. 1978. Grouping observations in digital simulation, *Management Science* 24(5), pp. 510–521.
- Ford, D. 2010. *Semiconductor industry set for highest annual growth in 10 years* [Online]. Available from: <http://tinyurl.com/3aenjzp> [Accessed 26 January 2010].
- Fowler, J., Brown, S., Gold, H. and Schoeming, A. 1997. Measurable improvements in cycle-time-constrained capacity, *Proceedings of the IEEE International Symposium on Semiconductor Manufacturing Conference*, pp. A21–A24.
- Fowler, J. and Robinson, J. 1995. Measurement and improvement of manufacturing capacity (MIMAC) project final report, *Technical report*, Sematech. Available from: <http://www.fabtime.com/files/MIMFINL.PDF> [Accessed 25 April 1996].
- Fowler, J. W., Leach, S. E., Mackulak, G. T. and Nelson, B. L. 2008. Variance-based sampling for simulating cycle timethroughput curves using simulation-based estimates, *Journal of Simulation* 2, pp. 69–80.
- Fowler, J. W., Park, S., MacKulak, G. T. and Shunk, D. L. 2001. Efficient cycle time-throughput curve generation using a fixed sample size procedure, *International Journal of Production Research* 39(12), pp. 2595–2613.
- Fowler, J. W. and Rose, O. 2004. Grand challenges in modeling and simulation of complex manufacturing systems, *Simulation* 80(9), pp. 469–476.

-
- Gafarian, A. V., Ancker Jr., C. J. and Morisaku, T. 1978. Evaluation of commonly used rules for detecting “steady state” in computer simulation, *Naval Research Logistics* 25(3), pp. 511–529.
- Gass, S. I. 1984. Documenting a computer-based model, *Interfaces* 14(3), pp. 84–93.
- Goldratt, E. M. 1990. *What is this thing called Theory of Constraints and how should it be implemented?*, North River Press.
- Goldratt, E. M. 1992. *The Goal: A Process of Ongoing Improvement*, North River Press.
- Gordon, G. 1977. *System Simulation*, Prentice Hall PTR, Upper Saddle River, NJ, USA.
- Gross, D. and Harris, C. M. 2003. *Fundamentals of Queuing Theory*, John Wiley and Sons, Singapore.
- Hoad, K., Robinson, S. and Davies, R. 2008. Automating warm up length estimation, *Proceedings of the 2008 Winter Simulation Conference*, pp. 532–540.
- Hoad, K., Robinson, S. and Davies, R. 2009. Automating discrete event simulation output analysis, *Proceedings of the 2009 INFORMS Simulation Society Research Workshop*, pp. 75–79.
- Hopp, W. J. and Spearman, M. L. 2001. *Factory Physics: Foundations of Manufacturing Management*, Irwin McGraw-Hill, Boston.
- Hopp, W. J., Spearman, M. L., Chayet, S., Donohue, K. L. and Gel, E. S. 2002. Using an optimized queueing network model to support wafer fab design, *IIE Transactions* 34(2), pp. 119–130.
- Hutcheson, G. D. 2000. Economics of semiconductor manufacturing, in Y. Nishi and R. Doering (eds), *Handbook of Semiconductor Manufacturing Technology*, Dekker, Chapter 37, pp. 1123–1139.
- Ignizio, J. P. 2009. *Optimizing Factory Performance: Cost-Effective Ways to Achieve Significant and Sustainable Improvement*, McGraw-Hill.
- Jacobs, J. H., Etman, L. F. P., Rooda, J. E. and Van Campen, E. J. J. 2001. Quantifying operational time variability: the missing parameter for cycle time reduction, *Proceedings of the 2001 IEEE/SEMI Advanced Semiconductor Manufacturing Conference*, pp. 1–10.

-
- Jacobs, J. H., Etman, L. F. P., van Campen, E. J. J. and Rooda, J. E. 2003. Characterization of operational time variability using effective process times, *IEEE Transactions on Semiconductor Manufacturing* 16(3), pp. 511–520.
- Jacobs, P. A. 1980. Heavy traffic results for single-server queues with dependent (EARMA) service and interarrival times, *Advances in Applied Probability* 12(2), pp. 517–529.
- Jeong, K., Wu, L. and Hong, J. 2009. IDEF method-based simulation model design and development framework, *Journal of Industrial Engineering and Management* 2(2), pp. 337–359.
- Johansson, B. and Grunberg, T. 2001. An enhanced methodology for reducing time consumption in discrete event simulation projects, *Proceedings of the 13th European Simulation Symposium*.
- Johnson, R. T., Feng, Y., Ankenman, B. E. and Nelson, B. L. 2004. Nonlinear regression fits for simulated cycle time vs. throughput curves for semiconductor manufacturing, *Proceedings of the 2004 Winter Simulation Conference*, Vol. 2, pp. 1951–1955.
- Johnson, R. T., Leach, S. E., Fowler, J. W. and Mackulak, G. T. 2004. Variance-based sampling for cycle time-throughput confidence intervals, *Proceedings of the 2004 Winter Simulation Conference*, pp. 720–725.
- Juang, J. and Huang, H. 2000. Queueing network analysis for an IC foundry, *IEEE Conference on Robotics and Automation*, Vol. 4, pp. 3389–3394.
- Kelton, W. D. 1980. *The startup problem in discrete-event simulation*, PhD thesis, University of Wisconsin.
- Kingman, J. F. C. 1966. On the algebra of queues, *Journal of Applied Probability* 3(2), pp. 285–326.
- Klein, M. and Kalir, A. 2006. A full factory transient simulation model for the analysis of expected performance in a transition period, *Proceedings of the 2006 Winter Simulation Conference*, pp. 1836–1839.
- Koo, P.-H., Jang, J. and Suh, J. 2005. Vehicle dispatching for highly loaded semiconductor production considering bottleneck machines first, *International Journal of Flexible Manufacturing Systems* 17, pp. 23–38.
- Kotcher, B. and Lumileds, P. 2011. Queueing models for wafer fabs, *FabTime Cycle Time Management Newsletter* 12(3), pp. 5–9.

-
- Kumar, P. 1994. Scheduling semiconductor manufacturing plants, *Control systems magazine* 14(6), pp. 1–23.
- Kwon, O. H. 2007. Perspective of the future semiconductor industry: Challenges and solutions, *Proceedings of the 44th ACM/IEEE Design Automation Conference*, p. xii.
- Law, A. 2008. How to build valid and credible simulation models, *Proceedings of the 2008 Winter Simulation Conference*, pp. 39–47.
- Law, A. M. and Carson, J. S. 1979. A sequential procedure for determining the length of a steady-state simulation, *Operations Research* 27(5), pp. 1011–1025.
- Law, A. M. and Kelton, W. D. 1997. *Simulation Modeling and Analysis*, McGraw-Hill Higher Education.
- Law, A. M. and McComas, M. G. 1991. Secrets of successful simulation studies, *Proceedings of the 1991 Winter Simulation Conference*, pp. 21–27.
- Lee, H. W. and Kim, T. H. 2005. A queueing model for multi-product production system under varying manufacturing environment, *Computational Science and Its Applications*, Vol. 3483 of *Lecture Notes in Computer Science*, Springer Berlin / Heidelberg, pp. 509–518.
- Li, J., Meerkov, S. and Zhang, L. 2009. Production Systems Engineering: A brief overview, *Stochastic Models of Manufacturing and Service Operations*, pp. 63–72.
- Li, N., Zhang, L., Zhang, M. and Zheng, L. 2005. Applied factory physics study on semiconductor assembly and test manufacturing, *Proceedings of the 2005 IEEE International Symposium on Semiconductor Manufacturing*, pp. 307–310.
- Li, N., Zhang, M. T., Deng, S., Lee, Z.-H., Zhang, L. and Zheng, L. 2007. Single-station performance evaluation and improvement in semiconductor manufacturing: A graphical approach, *International Journal of Production Economics* 107(2), pp. 397–403.
- Lopez, P., Terry, A., Daniely, D. and Kalir, A. 2005. Effective utilization (U_e) - a breakthrough performance indicator for machine efficiency improvement, *Proceedings of the 2005 IEEE International Symposium on Semiconductor Manufacturing*, pp. 63–66.
- Mackulak, G., Lawrence, F. and Colvin, T. 1998. Effective simulation model reuse: a case study for AMHS modeling, *Proceedings of the 1998 Winter Simulation Conference*, Vol. 2, pp. 979–984.

-
- Mahajan, P. and Ingalls, R. 2004. Evaluation of methods used to detect warm-up period in steady state simulation, *Proceedings of the 2004 Winter Simulation Conference*, Vol. 1, pp. 663–671.
- Martin, D. 1996. Optimizing manufacturing asset utilization, *Proceedings of the 1996 IEEE/SEMI Advanced Semiconductor Manufacturing Conference and Workshop*.
- MASM Lab Factory Datasets 1996. Available from: http://wwwalt.sim.uni-hannover.de/~svs/wise0910/pds/masmlab/factory_datasets/ [Accessed 12 January 2010].
- May, G. and Sze, S. 2004. *Fundamentals of Semiconductor Manufacturing*, Wiley.
- McIntosh, S. 1997. Conquering semiconductor’s economic challenges by increasing capital efficiency, *Proceedings of the 1997 IEEE International Symposium on Semiconductor Manufacturing Conference*, pp. 1–3.
- Miller, D. J. 1990. Simulation of a semiconductor manufacturing line, *Communications of the ACM* 33(10), pp. 98–108.
- Miltenburg, J., Cheng, C. H. and Yan, H. 2002. Analysis of wafer fabrication facilities using four variations of the open queueing network decomposition model, *IIE Transactions* 34(3), pp. 263–272.
- Montgomery, D. C. 1991. *Design and Analysis of Experiments*, 3rd edn, Wiley.
- Moore, G. E. 1995. Lithography and the future of Moore’s law, in R. D. Allen (ed.), *Advances in Resist Technology and Processing XII*, Vol. 2438, SPIE, pp. 2–17.
- Moore, G. E. 2000. Cramming more components onto integrated circuits, *Electronics* 1, pp. 56–59.
- Morrison, J. and Martin, D. 2007. Practical extensions to cycle time approximations for the $G/G/m$ -queue with applications, *IEEE Transactions on Automation Science and Engineering* 4(4), pp. 523–532.
- Mueller, R., Alexopoulos, C. and McGinnis, L. 2007. Automatic generation of simulation models for semiconductor manufacturing, *Proceedings of the 2007 Winter Simulation Conference*, pp. 648–657.
- Nordgren, W. 1995. Steps for proper simulation project management, *Proceedings of the 1995 Winter Simulation Conference*, pp. 68–73.

-
- Olhager, J. and Persson, F. 2008. Using simulation-generated operating characteristics curves for manufacturing improvement, in T. Koch (ed.), *Lean Business Systems and Beyond*, Vol. 257 of *IFIP Advances in Information and Communication Technology*, Springer Boston, pp. 195–204.
- Otto, K. 2007. *Anderson-darling normality test calculator* [Online]. Available from: www.robuststrategy.com/Software/ [Accessed 22 September 2008].
- Parkin, T. 2010. *Python community website* [Online]. Available from: <http://www.python.org/> [Accessed 17 August 2010].
- Paul, R. and Taylor, S. 2002. What use is model reuse: is there a crook at the end of the rainbow?, *Proceedings of the 2002 Winter Simulation Conference*, Vol. 1, pp. 648–652.
- Pidd, M. 1992. *Computer Simulation in Management Science*, John Wiley & Sons.
- Pidd, M. 2002. Simulation software and model reuse: a polemic, *Proceedings of the 2002 Winter Simulation Conference*, Vol. 1, pp. 772–775.
- Poeter, D. and Hachman, M. 2011. *Next Intel chips will have the world's first '3D' transistors* [Online]. Available from: <http://www.pcmag.com/article2/0,2817,2384897,00.asp#fbid=qI2QX9xoq07> [Accessed 12 October 2011].
- Potti, K. and Whitaker, M. 2003. Cycle time reduction at a major Texas Instruments wafer fab, *Proceedings of the 2003 IEEE/SEMI Advanced Semiconductor Manufacturing Conference*, pp. 106–110.
- Riddick, F. and Lee, Y. 2008. Representing layout information in the CMSD specification, *Proceedings of the 2008 Winter Simulation Conference*, pp. 1777–1784.
- Robinson, S. 1997. Simulation model verification and validation: Increasing the users' confidence, *Proceedings of the 1997 Winter Simulation Conference*, pp. 53–59.
- Robinson, S. 2002. A statistical process control approach for estimating the warm-up period, *Proceedings of the 2002 Winter Simulation Conference*, pp. 439–446.
- Robinson, S. 2003. *Simulation: The Practice of Model Development and Use*, Wiley.
- Robinson, S. 2005. Automated analysis of simulation output data, *Proceedings of the 2005 Winter Simulation Conference*, pp. 763–770.
- Robinson, S. 2006. Conceptual modeling for simulation: issues and research requirements, *Proceedings of the 38th Winter Simulation Conference*, pp. 792–800.

- Robinson, S. 2007. A statistical process control approach to selecting a warm-up period for a discrete-event simulation, *European Journal of Operational Research* 176(1), pp. 332–346.
- Robinson, S. and Bhatia, V. 1995. Secrets of successful simulation projects, *Proceedings of the 1995 Winter Simulation Conference*, pp. 61–67.
- Robinson, S. and Brooks, R. (eds) 2010. *Conceptual Modeling for Discrete-Event Simulation*, CRC Press.
- Rosenblueth, A. and Wiener, N. 1945. The role of models in science, *Philosophy of Science* 12(4), pp. 316–321.
- Rubinstein, R. and Melamed, B. 1998. *Modern Simulation and Modeling*, John Wiley & Sons.
- Rupp, K. and Selberherr, S. 2010. The economic limit to Moore’s law, *Proceedings of the IEEE* 98(3), pp. 351–353.
- Sadowski, D. and Grabau, M. 2004. Tips for successful practice of simulation, *Proceedings of the 2004 Winter Simulation Conference*, Vol. 1, pp. 56–61.
- Sargent, R. 1998. Verification and validation of simulation models, *Proceedings of the 1998 Winter Simulation Conference*, Vol. 1, pp. 121–130.
- Sattler, L. 1996. Using queueing curve approximations in a fab to determine productivity improvements, *Proceedings of the 1996 IEEE/SEMI Advanced Semiconductor Manufacturing Conference and Workshop*, pp. 140–145.
- Schaller, R. 1997. Moore’s law: Past, present and future, *IEEE Spectrum* 34(6), pp. 52–59.
- Schriber, T. and Brunner, D. 2010. Inside discrete-event simulation software: How it works and why it matters?, *Proceedings of the 2010 Winter Simulation Conference*, pp. 151–165.
- Semiconductor Industry Association Factsheet* 2010. Available from: <http://www.sia-online.org/> [Accessed 3 October 2010].
- Shanthikumar, J., Ding, S. and Zhang, M. 2007. Queueing theory for semiconductor manufacturing systems: A survey and open problems, *IEEE Transactions on Automation Science and Engineering* 4(4), pp. 513–522.

-
- Simon, W. 2008. *CPU transistor counts 1971-2008 and Moore's law* [Online]. Available from: http://commons.wikimedia.org/wiki/File:Moores_law.svg [Accessed 24 April 2008].
- Slifker, J. F. and Shapiro, S. S. 1980. The johnson system: Selection and parameter estimation, *Technometrics* 22(2), pp. 239–246.
- Sprenger, R. and Rose, O. 2010. *On the Simplification of Semiconductor Wafer Factory Simulation Models*, CRC Press, Chapter 17, pp. 451–470.
- Steele, M., Mollaghasemi, M., Rabadi, G. and Cates, G. 2002. Generic simulation models of reusable launch vehicles, *Proceeding of the 2002 Winter Simulation Conference* pp. 747–753.
- Storer, R. H., Swain, J. J., Venkatraman, S. and Wilson, J. R. 1988. Comparison methods for fitting data using Johnson translation distributions, *Proceedings of the 1988 Winter Simulation Conference*, pp. 476–481.
- Stroud, K. 1995. *Engineering Mathematics*, MacMillan.
- Taha, H. A. 2005. *Operations Research: An Introduction*, 8th edn, Prentice Hall.
- Valentin, E. and Verbraeck, A. 2002. Guidelines for designing simulation building blocks, *Proceedings of the 2002 Winter Simulation Conference*, Vol. 1, pp. 563–571.
- Veeger, C., Etman, L., van Herk, J. and Rooda, J. 2008. Generating cycle time-throughput curves using effective process time based aggregate modeling, *Proceedings of the 2008 IEEE/SEMI Advanced Semiconductor Manufacturing Conference*, pp. 127–133.
- Verbraeck, A. and Valentin, E. 2008. Design guidelines for simulation building blocks, *Proceedings of the 2002 Winter Simulation Conference*, pp. 923–932.
- Vucurevich, T. 2008. 3-D semiconductors: More from Moore, *Proceedings of the 45th Annual ACM/IEEE Design Automation Conference*, pp. 664–664.
- Wheeler, R. E. 1980. Quantile estimators of Johnson curve parameters, *Biometrika* 67(3), pp. 725–728.
- Whitner, R. and Balci, O. 1989. Guidelines for selecting and using simulation model verification techniques, *Proceedings of the 1989 Winter Simulation Conference*, pp. 559–568.

- Whitt, W. 1983. The queueing network analyzer, *The Bell Systems Technical Journal* 62(9), pp. 2779–2815.
- Whitt, W. 1989a. Planning queueing simulations, *Journal of Management Science* 35(11), pp. 1341–1366.
- Whitt, W. 1989b. Simulation run length planning, *Proceedings of the 1989 Winter Simulation Conference*, pp. 106–112.
- Whitt, W. 1993. Approximations for the GI/G/m queue, *Production and Operations Management* 2(2), pp. 114–161.
- Worldwide Sales of Semiconductors in Billion USD* 2010. Available from: <http://www.semi.org/> [Accessed 3 October 2010].
- Yao, X., Fernandez-Gaucherand, E., Fu, M. and Marcus, S. 2004. Optimal preventive maintenance scheduling in semiconductor manufacturing, *IEEE Transactions on Semiconductor Manufacturing* 17(3), pp. 345–356.
- Youngblood, S. 2006. *VV&A recommended practices guide* [Online]. Available from: <http://vva.msco.mil/> [Accessed 19 January 2011].

Appendices

Coded Algorithms for Designing DES Experiments

This chapter lists the coded algorithms used to create the framework for designing discrete event simulation experiments as described in Chapter 3.

A.1 Required number of simulation replications

This function returns the number of replication based on the precision of a confidence interval. If the number of required replications is greater than 30 then the central limit theorem applies and returns the value 30 (see Section 3.2 for discussion).

```
from math import sqrt
from scipy.stats import t.isf

def numRepsReq(data, relError=0.1, precision=0.05):
    xbar=mean(data)
    S_squared=var(data)
    adjError=relError/(1+relError)
```

```

repsReq=len(data)
tInv=(t.isf(precision, repsReq-1)*sqrt(S_squared/repsReq))/xbar

while tInv>adjError and repsReq<=30:
    repsReq=repsReq+1
    tInv=(t.isf(precision, repsReq-1)*sqrt(S_squared/repsReq))/xbar

return repsReq

```

A.2 Whitt approximation for simulation run length

Returns the approximate run length for a simple single stage queueing system according to Whitt (1989a) (see Section 3.3). Default values give an M/M/1 queue with mean process rate of 1.0, and the default confidence and relative confidence width are 0.95% and 0.05 respectively. A factor of safety can also be deployed (default value is 1.0) and the recommended minimum traffic intensity (utilisation) is 0.5. Note also, that the return value is rounded upwards to the nearest 1,000.

```

def whittRunLength(u,m,arrRate,te=1.0,scv_e=1.0,scv_a=1.0,c1=0.95,cw=0.05,FoS=1.0,min_u=0.3):
    z_val=ltqnorm(1-((1-c1)/2))
    if u<min_u:
        u=min_u
    numCust=(8*te*(scv_e+scv_a)*(z_val**2))/((u**2)*m*(cw**2)*((1-u)**2))
    return FoS*int(round((numCust/arrRate),-3))

```

Note: The function `whittRunLength` uses the function `ltqnorm` given in Appendix A.5.

A.3 Batch size approximation

This function returns the recommended number of batches according to the Von Neumann test for correlation and the Anderson Darling test for normality. It is based on an algorithm described in Section 3.4.1. The function has a terminating minimum batch size of 20.

```

def recNumBatches(data,precision=0.05,minBatches=20):
    batchSize =2
    n=len(data)
    numBatches=n/batchSize
    batch=makeBatch(data,numBatches)
    VN=vonNeumann(batch,precision)
    AD=andDar(batch,precision)

```



```

while bSize<=n/2:
    if VN == True or AD == False: # data is correlated or not normal
        "false test"
        bSize=bSize*2
        numBatches=n/bSize
        batch=makeBatch(data,numBatches)
        VN=vonNeumann(batch,precision)
        AD=andDar(batch,precision)
    else: # data is not correlated and is normal
        "true test"
    if bSize<=n/2:
        upperBound=bSize
        lowerBound=bSize/2
        break
    else:
        "an error occurred"

while (upperBound-lowerBound)>3:
    bSize = lowerBound + ((upperBound - lowerBound) / 2)
    numBatches=n/bSize
    batch=makeBatch(data,numBatches)
    VN=vonNeumann(batch,precision)
    AD=andDar(batch,precision)
    if VN == True or AD == False: # data is correlated or not normal
        "false test"
        lowerBound=bSize
    else:
        "true test"
        upperBound=bSize

if numBatches>minBatches:
    return numBatches
else:
    return minBatches

```

Note: The function `recNumBatches` uses the functions `vonNeumann` and `andDar` as follows.

A.3.1 Von Neumann algorithm

This algorithm returns TRUE if the input data is correlated according to Von Neumann test for autocorrelation.

```

def vonNeumann(data,alpha=0.05):
    n=len(data)
    temp1=[]
    for i in range(n-1):
        temp1.append((data[i]-data[i+1])**2)
    temp1=sum(temp1)

    ybar=mean(data)
    temp2=[]
    for y in data:
        temp2.append((y-ybar)**2)
    temp2=sum(temp2)

    VNstat=sqrt(((n**2)-1)/(n-2))*(1-(temp1/(2*temp2)))
    z=ltqnorm(1-alpha)

    if VNstat>z:
        return True # data is correlated
    else:
        return False # data is not correlated

```

Note: The function `vonNeumann` uses the function `ltqnorm` (see Appendix A.5).

A.3.2 Anderson-Darling test for normality

The `andDar` function returns `TRUE` if the data series is normally distributed. The function was adapted from Otto (2007).

```
def andDar(data,alpha=0.05):
    n=len(data)
    ybar=mean(data)
    y=std(data,ddof=1)

    s_data=sort(data)

    F1i=[]
    for i in s_data:
        F1i.append(norm.cdf((i-ybar)/y))

    F2i=[]
    for i in F1i:
        F2i.append(1-i)

    F2i_s=sort(F2i)
    Si=[]
    for i in range(len(F2i)):
        Si.append(((2*(i+1))-1)*(log(F1i[i])+log(F2i_s[i])))

    ADTestStat=((-1)*(sum(Si))/n)-n
    ADStarTestStat=ADTestStat*(1+(0.75/n)+(2.25/(n**2)))

    if ADStarTestStat>=0.6 and ADStarTestStat < 13:
        p1 = exp(1.2937-(5.709*ADStarTestStat)+(0.0186*(ADStarTestStat**2)))
    else:
        p1 = 0

    if ADStarTestStat< 0.6 and ADStarTestStat >= 0.34:
        p2 = exp(0.9177 - (4.279*ADStarTestStat)-(1.38*(ADStarTestStat**2)))
    else:
        p2 = 0

    if ADStarTestStat<0.34 and ADStarTestStat>=0.2:
        p3 = exp((( -1)*8.318)+(42.796*ADStarTestStat)-(59.938*(ADStarTestStat**2)))
    else:
        p3 = 0

    if ADStarTestStat < 0.2:
        p4 =1- exp((( -1)*13.436)+(101.14*ADStarTestStat)-(223.73*(ADStarTestStat**2)))
    else:
        p4 = 0

    maxp=max(p1,p2,p3,p4)
    if maxp>alpha: # then the data is normally distributed
        return True
    else:
        return False
```

Note: The function `andDar` uses the function `tlqnorm` (see Appendix A.5).

A.4 SPC Algorithm

The function `SPCMethod` returns the index number of the element in the time series where steady state is assumed to have begun according to the statistical process control (SPC) control rules (Robinson, 2007). The warm-up period for the model can be selected by identifying the point at which the time-series data is in control and remains in control.

The function returns the value -1 if data failed on last half of data. If the series is always in control, the warm up period is assumed to be the point where the mean line is crossed for the first time by the transient. See Section 3.4.1 for more detailed discussion.

```

## Declare Imports
from numpy import *
from scipy.stats import t,norm
import pylab

##SPC Function
def SPCMethod(data,plotit=False):
    """dataArray should be of the form numPyarray ([[rep1],[rep2],...,[rep n]])"""

    n=data.shape[0]
    m=data.shape[1]
    mean_vals=zeros(m)

    for i in range(m):
        sum=0.0
        for j in range(n):
            sum=sum+data[j,i]
        mean_vals[i]=sum/n

    b=recNumBatches(mean_vals)
    varArray=batchVar(data,b)
    batchArray=makeBatch(mean_vals,b)

    p0=b/2+1 # p0 is start of second half of data
    sum=0.0
    for i in arange(p0,b,1):
        sum=sum+batchArray[i]
    mean_val=sum/(b-p0-1)

    sum=0.0
    for i in arange(p0,b,1):
        sum=sum+varArray[i]
    std_val=sqrt(sum/(b/2))

    # control limits
    UL3 = mean_val + ((3 * std_val) / sqrt(n))
    UL2 = mean_val + ((2 * std_val) / sqrt(n))
    UL1 = mean_val + ((1 * std_val) / sqrt(n))
    LL1 = mean_val - ((1 * std_val) / sqrt(n))
    LL2 = mean_val - ((2 * std_val) / sqrt(n))
    LL3 = mean_val - ((3 * std_val) / sqrt(n))

    # Test 1: a point plots outside a 3 sigma control limit
    test1=0
    for i in arange(b-1,-1,-1):
        if batchArray[i]>UL3 or batchArray[i]<LL3:
            test1=i
            break

    # Test 2: two out of three consecutive points plot outside 2 sigma control limit
    test2=0
    for i in arange(b-1,2,-1):
        if batchArray[i]>UL2:
            if batchArray[i-1]>UL2 or batchArray[i-2]>UL2:
                test2 = i
                break
            elif batchArray[i]<LL2:
                if batchArray[i-1]<LL2 or batchArray[i-2]<LL2:
                    test2 = i
                    break

    # Test 3: four out of five consecutive points plot outside the a 1-sigma control
    test3=0
    for i in arange(b-1,4,-1):
        if batchArray[i]>UL1:
            failCount=1
            for j in arange(i-1,i-4,-1):
                if batchArray[j]>UL1:
                    failCount=failCount+1
            if failCount>3:
                test3=i
                break

```

```

elif batchArray[i]<LL1:
    failCount=1
    for j in arange(i-1,i-4,-1):
        if batchArray[j]<LL1:
            failCount=failCount+1
if failCount>3:
    test3=i
    break

# Test 4: eight consecutive points plot on one side of the mean
test4=0
for i in arange(b-1,7,-1):
    if batchArray[i]>mean_val:
        failCount=1
        for j in arange(i-1,i-7,-1):
            if batchArray[j]>mean_val:
                failCount=failCount+1
    if failCount>7:
        test4=i
        break
    elif batchArray[i]<mean_val:
        failCount=1
        for j in arange(i-1,i-7,-1):
            if batchArray[j]<mean_val:
                failCount=failCount+1
    if failCount>7:
        test4=i
        break

# plot
if plotit == True:
    pylab.plot(batchArray,label='transient',color='black',linewidth=2)
    pylab.plot(ones(b)*UL3,label='spc limit 3',color='black',linewidth=0.5,linestyle='dotted'
    )
    pylab.plot(ones(b)*UL2,label='spc limit 2',color='black',linewidth=0.5,linestyle='-.')
    pylab.plot(ones(b)*UL1,label='spc limit 1',color='black',linewidth=0.5,linestyle='dashed'
    )
    pylab.plot(ones(b)*LL1,color='black',linewidth=0.5,linestyle='dashed')
    pylab.plot(ones(b)*LL2,color='black',linewidth=0.5,linestyle='-.')
    pylab.plot(ones(b)*LL3,color='black',linewidth=0.5,linestyle='dotted')
    pylab.plot(ones(b)*mean_val,label='mean',color='black')
    pylab.legend(bbox_to_anchor=(1.05, 1), loc=2)
    pylab.xlabel('time series')
    pylab.ylabel('performance indicator')
    pylab.title('Warm up period using SPC Method')
    pylab.grid(False)
    pylab.savefig('spc_plot.ps')
    pylab.show()

if max(test1,test2,test3,test4)>p0: # test failed on last half of data
    return -1
elif max(test1,test2,test3,test4)>0: # test failed in first half of data
    return max(test1,test2,test3,test4)+1
else:
    # assume warmup occurs at point where transient crosses mean
    if batchArray[i]<mean_val:
        for i in arange(2,b,1):
            if batchArray[i]>mean_val:
                return i
        else:
            for i in arange(2,b,1):
                if batchArray[i]<mean_val:
                    return i

```

Note: The function SPCMethod uses the python module Pylab, a tool for plotting data in python scripts. Plotting can be turned off in the function by setting the input variable `plotit = FALSE`.

A.5 Miscellaneous Functions

This section lists the miscellaneous functions and modules used to code the framework described in Chapter 3.

A.5.1 Inverse normal distribution function

This function returns an approximation of the inverse cumulative standard normal distribution function, i.e., given P , it returns an approximation to the X satisfying $P = Pr\{Z \leq X\}$ where Z is a random variable from the standard normal distribution based on an algorithm provided by (Acklam, 2003).

```
def ltqnorm( p ):
    if p <= 0 or p >= 1:
        raise ValueError("Argument to ltqnorm %f must be in open interval (0,1)"% p)

    # Coefficients in rational approximations.
    a = (-3.969683028665376e+01, 2.209460984245205e+02, \
         -2.759285104469687e+02, 1.383577518672690e+02, \
         -3.066479806614716e+01, 2.506628277459239e+00)
    b = (-5.447609879822406e+01, 1.615858368580409e+02, \
         -1.556989798598866e+02, 6.680131188771972e+01, \
         -1.328068155288572e+01)
    c = (-7.784894002430293e-03, -3.223964580411365e-01, \
         -2.400758277161838e+00, -2.549732539343734e+00, \
         4.374664141464968e+00, 2.938163982698783e+00)
    d = ( 7.784695709041462e-03, 3.224671290700398e-01, \
         2.445134137142996e+00, 3.754408661907416e+00)

    # Define break-points.
    plow = 0.02425
    phigh = 1 - plow

    # Rational approximation for lower region:
    if p < plow:
        q = sqrt(-2*log(p))
        return (((c[0]*q+c[1])*q+c[2])*q+c[3])*q+c[4])*q+c[5] / \
               (((d[0]*q+d[1])*q+d[2])*q+d[3])*q+1)

    # Rational approximation for upper region:
    if phigh < p:
        q = sqrt(-2*log(1-p))
        return -((((c[0]*q+c[1])*q+c[2])*q+c[3])*q+c[4])*q+c[5] / \
                (((d[0]*q+d[1])*q+d[2])*q+d[3])*q+1)

    # Rational approximation for central region:
    q = p - 0.5
    r = q*q
    return (((((a[0]*r+a[1])*r+a[2])*r+a[3])*r+a[4])*r+a[5])*q / \
            (((b[0]*r+b[1])*r+b[2])*r+b[3])*r+b[4])*r+1)
```

A.5.2 Batch means method

Returns the batch means series based on a requested number of batches (Law and Carson, 1979; Law and Kelton, 1997). See Section 3.4.1 for details.

```

def makeBatch(data,b):
    n = len(data)
    b_width = n/b
    output = []

    for i in range(b):
        sum = 0.0
        for j in arange((b_width*i),b_width*(i+1),1):
            sum = sum + data[j]
        if i == (b-1) and b_width*(i+1) < (n-1): # if last batch and some odd points are left
            for k in arange(b_width*(i+1),n,1):
                sum = sum + data[k]
            output.append(sum/(b_width + len(rem_range)))
        else:
            output.append(sum/b_width)

    return output

```

A.5.3 Batch variance

The function `batchVar` calculates the batch variance given a set of time series (from individual replications) and a specified batch size. It returns an array of the batch variance for each batch across the reflections. If the batch means of a replication are given by,

$$\bar{Y}_i = \frac{\sum_{i=(b-1)k+1}^{bk} Y_i}{b}, \quad (\text{A.1})$$

for a number of batches b of batch width k and a data series Y_i , where $i = 1, 2, \dots, n$. Then the variance of the batch means across the replications $j = 1, 2, \dots, m$ for each batch is given by $Var(\bar{Y}_i)_j$.

```

def batchVar(data,b):
    numReps=data.shape[0]
    m=data.shape[1]
    b_width = m/b
    varArray=zeros(b)

    for i in range(b):
        means=zeros(numReps)
        for c in range(numReps):
            sum=0.0
            for j in arange((b_width*i),b_width*(i+1),1):
                sum=sum+data[c,j]
            if i==(b-1) and b_width*(i+1) < (m-1):
                for k in arange((b_width*(i+1)),m,1):
                    sum=sum+data[c,k]
                means[c]=sum/(b_width+len(rem_range))
            else:
                means[c]=sum/b_width
        varArray[i]=var(means)

    return varArray

```

A.5.4 Queue operating point

The function `optOperating` solves for the optimum operating point on a queueing curve approximation. See Section 3.1.2 for discussion.

```
from sympy import *  
  
def optOperating(te=1, cva2=1, cve2=1, m=1):  
    u=Symbol('u')  
    return solve((diff(u/(te*(1+((cva2+cve2)/2)*(u**sqrt(2*(m+1))-1))/(m*(1-u))), u))
```

Note: The function `optOperating` uses symbolic functions called from the third party SYMPY module for Python.

Flexible Toolset Modelling

Application Code

This chapter includes the program code and decision algorithms used to create the Flexible Toolset Modelling (FTM) application in Visual Basic (VB) and ExtendSim. Additional description is given in Chapter 4.

B.1 Front-End

The graphical user interface (GUI) home screen Run button executes the main function of the application after selection of the tools to be examined.

```
Sub DoProgram()  
    'This subroutine executes the main program and is triggered by the Run button in the GUI  
    Userform  
  
    If RetrieveModel(modelFile) = False Then  
        MsgBox "Error occurred opening model " + modelFile + " . in sub DoProgram."  
    End If
```



```

GetDatafromServers
CreateProcessDataSheet
getInterarrivalTimes
GetProcessTimes
FillToolOfflineData
GetDT
GetPM
ToolRank.Show
PassAllDataToExtendSim

If CustomPMFlag Then
    GetCustomPM
Else
    GetPM
End If

RunExtendSimModel
End Sub

```

```

Public Function GetTools()

    Dim i, y, r As Integer
    Dim rawTools() As Variant
    Dim delTool As Boolean
    Dim chr As Integer
    Dim tempString As String

    On Error GoTo ErrorHandler:

    y = Sheets("InfoSheet").Cells(1, 1).End(xlDown).Row
    r = 0
    For i = 1 To y
        If Sheets("InfoSheet").Cells(i, 2) = True Then
            r = r + 1
            ReDim Preserve rawTools(r)
            rawTools(r) = Sheets("InfoSheet").Cells(i, 1)
        End If
    Next i

    Validate:
    'Validate the data
    For i = 1 To UBound(rawTools)
        delTool = False
        If IsEmpty(rawTools(i)) = True Then
            delTool = True
            GoTo Delete:
        End If

        tempString = CStr(rawTools(i))
        For chr = 1 To Len(tempString)
            If Mid(tempString, chr, 1) = "/" Or Mid(tempString, chr, 1) = "(" Or Mid(tempString, chr,
                1) = ")" Then
                delTool = True
                GoTo Delete:
            End If
        Next
    Next

    Delete:
    If delTool = True Then
        j = i
        For j = i To UBound(rawTools) - 1
            rawTools(j) = rawTools(j + 1)
        Next j
        ReDim Preserve rawTools(UBound(rawTools) - 1)
        GoTo Validate:
    End If
    Next

    'Get unique items
    Tools = UniqueItems(rawTools, False)
    GetTools = 1

    ErrorHandler:
    If Err.Number <> 0 Then
        If Err.Number = 9 Then
            GetTools = -1
            Exit Function
        Else

```

```

    MsgBox Err.Number & ": " & Err.Description & " (" & Erl & ") "
    Exit Function
End If
End If
End Function

```

B.2 Data Collection and Generation of Distributions for Model

Procedures, algorithms and programming code associated with collection, sorting and filtration of raw data for the simulation model.

B.2.1 Data pull and cross-referencing

The CreateProcessDataSheet procedure takes the lot history and the entity/tool history and combines them into a single database. The data is then scrubbed by removing any test lot data, accounting for any rework lots and deleting any incomplete or error logs.

```

Sub CreateProcessDataSheet ()

    Dim i, j, k, y, y2, rwCount, count As Long
    Dim tempvall, tempj As Integer
    Dim rw As Variant
    Dim entHistVals() As Variant
    Dim foundFlag As Boolean

    Application.ScreenUpdating = False

    'Fill in the headings in ProcessData sheet
    With Sheets("ProcessData")
        .Cells.Clear 'clear the sheet
        .Cells(1, 1) = "ENTITY"
        .Cells(1, 2) = "LOT"
        .Cells(1, 3) = "OPERATION"
        .Cells(1, 4) = "PREVOUT_DATE"
        .Cells(1, 5) = "IN_DATE"
        .Cells(1, 6) = "BEGIN RUN"
        .Cells(1, 7) = "END PROCESS"
        .Cells(1, 8) = "END RUN"
        .Cells(1, 9) = "OUT_DATE"
        .Columns("D:I").NumberFormat = "dd/mm/yy hh:mm:ss"
        .Columns("D:I").ColumnWidth = 17
    End With

    'Copy the entries from lotHist
    y = Sheets("LotHist").Cells(1, 1).End(xlDown).Row
    For i = 2 To y
        If Sheets("LotHist").Cells(i, 1) <> "" Then
            With Sheets("ProcessData")
                .Cells(i, 1) = Sheets("LotHist").Cells(i, 1) 'Entity
                .Cells(i, 2) = Sheets("LotHist").Cells(i, 2) 'Lot
                .Cells(i, 3) = Sheets("LotHist").Cells(i, 3) 'Operation
            End With
        End If
    Next i
End Sub

```

Appendix B. FTM Application Code

```

        .Cells(i, 4) = Sheets("LotHist").Cells(i, 4)      'Prevout
        .Cells(i, 5) = Sheets("LotHist").Cells(i, 5)      'In Date
        .Cells(i, 9) = Sheets("LotHist").Cells(i, 6)      'OutDate
    End With
End If
Next i

'Remove test lots
y = Sheets("ProcessData").Cells(1, 1).End(xlDown).Row
On Error Resume Next
For i = y To 2 Step -1
    tempvall = (Mid(Sheets("ProcessData").Cells(i, 2), 8, 1))
    If Err.Number = 13 Then 'this means that the last digit is not an integer value
        Sheets("ProcessData").Cells(i, "A").EntireRow.Delete
        Err.Clear
    End If
Next i

'Sort the ProcessData by Entity, then by IN_Date
y = Sheets("ProcessData").Cells(1, 1).End(xlDown).Row
Sheets("ProcessData").Select
Sheets("ProcessData").Range("A2:i" & y).Select
Selection.Sort Key1:=Range("A1"), Order1:=xlAscending, Key2:=Range("E1"), Order2:=
    xlAscending

'Sort the EntHist by Entity, then by TXN Date
y2 = Sheets("EntHist").Cells(1, 1).End(xlDown).Row
Sheets("EntHist").Select
Sheets("EntHist").Range("A2:F" & y2).Select
Selection.Sort Key1:=Range("A1"), Order1:=xlAscending, Key2:=Range("B1"), Order2:=
    xlAscending

'Count the "BEGIN RUN" in EntHist
count = 0
For i = 2 To y2
    If Sheets("EntHist").Cells(i, 5) = "BEGIN RUN" Then
        count = count + 1
    End If
Next i

'Copy the EntHist values into a 4 column array with cols Tool Name, Begin Run, End Process,
End Run
ReDim entHistVals(count, 4)
i = 2
r = 0
Do Until i > y2
    If Sheets("EntHist").Cells(i, 5) = "BEGIN RUN" Then
        j = 1
        Do Until Sheets("EntHist").Cells(i + j, 5) = "END PROCESS" Or j > 20
            j = j + 1
        Loop
        k = 1
        Do Until Sheets("EntHist").Cells(i + j + k, 5) = "END RUN" Or k > 20
            k = k + 1
        Loop
        If j <= 20 And k <= 20 Then
            r = r + 1
            entHistVals(r, 1) = Sheets("EntHist").Cells(i, 1)
            entHistVals(r, 2) = Sheets("EntHist").Cells(i, 2)
            entHistVals(r, 3) = Sheets("EntHist").Cells(i + j, 2)
            entHistVals(r, 4) = Sheets("EntHist").Cells(i + j + k, 2)
            i = i + j + k + 1
        Else
            i = i + 1
        End If
    Else
        i = i + 1
    End If
Loop

'Match the array up with the LotHist values in ProcessData Sheet
foundFlag = False
For i = 2 To y
    If foundFlag = False Then
        j = tempj
    End If
    Do Until (Sheets("ProcessData").Cells(i, 1) = entHistVals(j, 1) And _
        (((Sheets("ProcessData").Cells(i, 5) - entHistVals(j, 2)) * 24) ^ 2) ^ 0.5) <= 0.15)
        Or _
        j >= count

```

```

    j = j + 1
Loop
If j < count Then
    If (((Sheets("ProcessData").Cells(i, 5) - entHistVals(j, 2)) * 24) ^ 2) ^ 0.5) <= 0.15
        And _
        (((Sheets("ProcessData").Cells(i, 9) - entHistVals(j, 4)) * 24) ^ 2) ^ 0.5) <= 0.15
            And _
            Sheets("ProcessData").Cells(i, 1) = entHistVals(j, 1) Then
                Sheets("ProcessData").Cells(i, 6) = entHistVals(j, 2)
                Sheets("ProcessData").Cells(i, 7) = entHistVals(j, 3)
                Sheets("ProcessData").Cells(i, 8) = entHistVals(j, 4)
                foundFlag = True
                tempj = j
            End If
        Else
            foundFlag = False
        End If
    End If
Next i

'Remove any zero date entries
For i = y To 2 Step -1
    delRow = False
    tempval = WorksheetFunction.CountA(Sheets("ProcessData").rows(i))
    If WorksheetFunction.CountA(Sheets("ProcessData").rows(i)) <> 9 Then
        delRow = True
    End If

    For j = 1 To 9
        If Sheets("ProcessData").Cells(i, j) = 0 Then
            delRow = True
        End If
    Next j

    If delRow = True Then
        Sheets("ProcessData").rows(i).EntireRow.Delete
    End If
Next i

'Sort the ProcessData by Entity, then opID then lot
y = Sheets("ProcessData").Cells(1, 1).End(xlDown).Row
Sheets("ProcessData").Select
Sheets("ProcessData").Range("A2:I" & y).Select
Selection.Sort Key1:=Range("A1"), Order1:=xlAscending, Key2:=Range("C1"), Order3:=
    xlAscending, Key2:=Range("B1"), Order3:=xlAscending

'Count Rework
rwCount = 0
For i = 2 To y - 1
    If Sheets("ProcessData").Cells(i, 1) = Sheets("ProcessData").Cells(i + 1, 1) And _
        Sheets("ProcessData").Cells(i, 2) = Sheets("ProcessData").Cells(i + 1, 2) And _
        Sheets("ProcessData").Cells(i, 3) = Sheets("ProcessData").Cells(i + 1, 3) Then
        rwCount = rwCount + 1
    End If
Next i

Application.ScreenUpdating = True
End Sub

```

B.2.2 Calculating arrival rates

The `getInterarrivalTimes` procedure calculates the inter-arrival mean for each operation passing through the selected tools or toolsets.

```

Sub getInterarrivalTimes()
    Dim i, j, k, y, count As Long
    Dim sum As Double
    Dim rangel As Range
    Dim percentOp() As Variant
    Dim opID() As Variant

```

```

Dim meanIA() As Variant

Application.ScreenUpdating = False
'Sort the data by operation then PREVOUT_DATE
y = Sheets("ProcessData").Cells(1, 1).End(xlDown).Row
Sheets("ProcessData").Select
Sheets("ProcessData").Range("A2:I" & y).Select
Selection.Sort Key1:=Range("C1"), Order1:=xlAscending, Key2:=Range("D1"), Order2:=
    xlAscending

'Populate opID
Set rangel = Sheets("ProcessData").Range("C2:C" & y)
opID = UniqueItems(rangel, False)
ReDim meanIA(UBound(opID))
ReDim percentOp(UBound(opID))

'Populate meanIA
For j = 1 To UBound(opID)
    count = 0
    sum = 0
    For i = 2 To (y - 1)
        If Sheets("ProcessData").Cells(i, 3) = opID(j) And _
            Sheets("ProcessData").Cells(i + 1, 3) = opID(j) And _
            Sheets("ProcessData").Cells(i, 4) <> "" And _
            Sheets("ProcessData").Cells(i + 1, 4) <> "" Then
            count = count + 1
            sum = sum + ((Sheets("ProcessData").Cells(i + 1, 4) - Sheets("ProcessData").Cells(i,
                4)) * 24)
        End If
    Next i
    percentOp(j) = count
    meanIA(j) = sum / count
Next j

'Populate percentOp
sum = 0
For j = 1 To UBound(percentOp)
    sum = sum + percentOp(j)
Next j

For j = 1 To UBound(percentOp)
    percentOp(j) = percentOp(j) / sum
Next j

'Populate Arrival Info, an array that holds opID, mean IA time and %op.
ReDim arrivalInfo(UBound(opID), 3)
For j = 1 To UBound(arrivalInfo)
    arrivalInfo(j, 1) = opID(j)
    arrivalInfo(j, 2) = meanIA(j)
    arrivalInfo(j, 3) = percentOp(j)
Next j

Application.ScreenUpdating = True
End Sub

```

B.2.3 Calculating process time

The `getProcessTimes` procedure collects data to construct the process pattern for each operation type that passes through the toolsets. The process time data is collected, filtered and passed into a subroutine `JSFitter` (see Appendix F.1) that outputs the Johnson distribution parameters.

```

Sub getProcessTimes()

Dim i, j, y, c, k, r, lastRow As Long
Dim rangel As Range

```

```

Dim processArray(), moveOutArray(), processArray2() As Double
Dim toolID() As Variant
Dim jValues() As Variant
Dim tempOpID() As Variant
Dim tempArray1(), tempArray2() As Variant
Dim meanPTforOpOnTool() As Variant

Sheets("ProcessData").Select
y = Sheets("ProcessData").Cells(1, 1).End(xlDown).Row
Sheets("ProcessData").Range("A2:I" & y).Select
Selection.Sort Key1:=Range("A1"), Order1:=xlAscending, Key2:=Range("C1"), Order2:=
xlAscending

toolID = Tools

'i and j are the lower and upper bounds of the tool in question
tempSum = 0
j = 2
For c = 1 To UBound(toolID)
    i = j
    Do Until Sheets("ProcessData").Cells(i, 1) = toolID(c)
        i = i + 1
    Loop
    j = i
    Do Until Sheets("ProcessData").Cells(j, 1) <> toolID(c)
        j = j + 1
    Loop

    Set rangel = Sheets("ProcessData").Range(Cells(i, 3), Cells(j, 3))
    tempOpID = UniqueItems(rangel, False) 'get unique operations for that tool
    For t = 1 To UBound(tempOpID)
        tempCount = 0
        For z = i To j
            If tempOpID(t) = Sheets("ProcessData").Cells(z, 3) Then
                tempCount = tempCount + 1 'the number times a unique operation was carried out on a
                tool
            End If
        Next z
        If tempCount > 30 Then
            tempSum = tempSum + 1
            ReDim Preserve tempArray1(tempSum)
            ReDim Preserve tempArray2(tempSum)
            tempArray1(tempSum) = toolID(c)
            tempArray2(tempSum) = tempOpID(t)
        End If
    Next t
Next c

ReDim pTimes(UBound(tempArray1), 10)
For i = 1 To UBound(pTimes)
    pTimes(i, 1) = tempArray1(i) 'the dummy tool list
    pTimes(i, 2) = 1 'the tool type 1: Single Process
    pTimes(i, 3) = tempArray2(i) ' the dummy op id list
Next i

ReDim meanPTforOpOnTool(UBound(pTimes))
ReDim scvPTforOpOnTool(UBound(pTimes))

'Build Arrays and pass for JSFitter
For c = 1 To UBound(pTimes)
    j = 0
    For i = 2 To y
        If pTimes(c, 1) = Sheets("ProcessData").Cells(i, 1) And _
            pTimes(c, 3) = Sheets("ProcessData").Cells(i, 3) Then
            j = j + 1
            ReDim Preserve processArray(j)
            ReDim Preserve moveOutArray(j)
            ReDim Preserve processArray2(j)
            processArray(j) = (Sheets("ProcessData").Cells(i, "H") - Sheets("ProcessData").Cells(i
            , "E")) * 24
            moveOutArray(j) = (Sheets("ProcessData").Cells(i, "I") - Sheets("ProcessData").Cells(i
            , "H")) * 24
            processArray2(j) = (Sheets("ProcessData").Cells(i, "I") - Sheets("ProcessData").Cells(
            i, "E")) * 24
        End If
    Next i

    meanPTforOpOnTool(c) = WorksheetFunction.Average(processArray2)
    scvPTforOpOnTool(c) = ((WorksheetFunction.StDev(processArray2)) ^ 2) / ((WorksheetFunction
    .Average(processArray2)) ^ 2)

'Fill process time parameters

```

```

jValues = JSFitter(processArray)
If jValues(1) > 0 Then
    pTimes(c, 4) = jValues(1) 'Dist Type
    pTimes(c, 5) = jValues(2) 'neta
    pTimes(c, 6) = jValues(3) 'gamma
    pTimes(c, 7) = jValues(4) 'lambda
    pTimes(c, 8) = jValues(5) 'epsilon
    pTimes(c, 9) = WorksheetFunction.Average(moveOutArray) 'move out average
    pTimes(c, 10) = j / y 'the percent of all operations
Else 'Insufficient data for JSFitter
    MsgBox ("Warning insufficient data for Johnson Distribution")
End If
Next c

'Rename pTimes ToolID column because Extend cant take strings
For i = 1 To UBound(pTimes)
    j = 1
    Do Until pTimes(i, 1) = toolID(j)
        j = j + 1
    Loop
    pTimes(i, 1) = j
Next i

'Create a meanPT() and an scvPT() array for the tools to be used in QTOpCurve subroutine
ReDim meanPT(UBound(toolID))
ReDim scvPT(UBound(toolID))

For c = 1 To UBound(toolID)
    count = 0
    tempPT = 0
    tempSCV = 0
    For i = 1 To UBound(pTimes)
        If pTimes(i, 1) = c Then
            count = count + 1
            tempPT = tempPT + meanPTforOpOnTool(i)
            tempSCV = tempSCV + scvPTforOpOnTool(i)
        End If
    Next i
    meanPT(c) = tempPT / count
    scvPT(c) = tempSCV / count
Next c
End Sub

```

B.2.4 Estimate downtime parameters

The fillToolOfflineData procedure collects all references to a down event from the tool history. The event name tags are also captured and the user is prompted to distinguish between unscheduled and scheduled downtime from the downtime tag names. Two additional functions, getDT and getPM then collect all the necessary data into arrays for distribution building.

```

Sub fillToolOfflineData()
    'This subroutine fills the array ToolOfflineData() with all references to either a PM event
    'or a downtime event

    Dim i, j, k, y, m, n, x As Long
    Dim pmFlag As Boolean
    Dim OfflineID As Integer
    Dim MinReqTime As Double
    Dim count1, count2 As Integer
    Dim ToolOfflineData() As Variant

    MinReqTime = 1 'This is the minimum time required between offline events for them to be
    unique

```

```

Sheets("EntHist").Select
y = Sheets("EntHist").Cells(1, 1).End(xlDown).Row
Sheets("EntHist").Range("A2:F" & y).Select
Selection.Sort Key1:=Range("A1"), Order1:=xlAscending, Key2:=Range("B1"), Order2:=
    xlAscending 'By Entity then Time

k = 0
For i = 2 To y
    If Sheets("EntHist").Cells(i, "C") = "D" Or Sheets("EntHist").Cells(i, "D") = "D" Then
        k = k + 1
    End If
Next i

'Fill all into ToolOfflineData array 1-6 as in wkbook, col 7 is sched flag, col 8 is unique
offline occurrence identifier
ReDim ToolOfflineData(k, 8)
ReDim dtStates(k)

'Fill ToolOfflineData with all references to a down state
k = 0
For i = 2 To y
    If Sheets("EntHist").Cells(i, "C") = "D" Or _
        Sheets("EntHist").Cells(i, "D") = "D" Then
        k = k + 1
        For j = 1 To 6
            ToolOfflineData(k, j) = Sheets("EntHist").Cells(i, j)
        Next j
        dtStates(k) = ToolOfflineData(k, 6)
    End If
Next i

dtStates = UniqueItems(dtStates, False) 'dtStates array now contains all unique DT states
Call SelectPMstates 'puts all pm event names into an array called PMstates

'Use this loop to mark the downtime event 1 for PM and 2 for unscheduled(DT)
OfflineID = 0
i = 1
Do Until i >= k - 1
    If ToolOfflineData(i, 4) = "D" And ToolOfflineData(i, 3) = " " Then 'It is the start of an
        offline event
        pmFlag = False
        For j = 1 To UBound(PMState)
            If ToolOfflineData(i, 6) = PMState(j) Or ToolOfflineData(i + 1, 6) = PMState(j) Then
                pmFlag = True 'it is a PM event
            Exit For
        End If
        Next j

        m = 1
        sameEvent:
        Do Until (ToolOfflineData(i + m, 4) = " " And ToolOfflineData(i + m, 3) = "D") Or ((i +
            m) >= k - 1) 'end of the DT event
            m = m + 1
        Loop

        n = 1
        Do Until (ToolOfflineData(i + m + n, 4) = "D" And ToolOfflineData(i + m + n, 3) = " ")
            Or ((i + m + n) >= k - 1) 'start of next DT event
            n = n + 1
        Loop

        'check exit
        If (i + m + n) >= k - 1 Then 'we are finished
            Exit Do
        End If

        'Check if its the same event occurrence
        If ToolOfflineData(i, 1) = ToolOfflineData(i + m, 1) And _
            ToolOfflineData(i, 1) = ToolOfflineData(i + m + n, 1) Then 'all the same tool
            If ((ToolOfflineData(i + m + n, 2) - ToolOfflineData(i + m, 2)) * 24) <= MinReqTime
                Then 'they are the same event
                m = m + n
                GoTo sameEvent:
            End If
        End If

        If ToolOfflineData(i, 1) = ToolOfflineData(i + m, 1) Then
            OfflineID = OfflineID + 1
            For z = i To (i + m)
                ToolOfflineData(z, 8) = OfflineID
            If pmFlag = True Then
                ToolOfflineData(z, 7) = 1 'it is a PM

```



```

        Else
            ToolOfflineData(z, 7) = 2 'it is an unsched down
        End If
    Next z
    i = i + m + 1
    GoTo QuickLoop:
End If
End If
i = i + 1
QuickLoop:
Loop

'Count the number of PM events so we can split the array into two arrays rawPMdata and
rawDTdata
count1 = 0
count2 = 0
For i = 1 To UBound(ToolOfflineData)
    If ToolOfflineData(i, 7) = 1 And IsEmpty(ToolOfflineData(i, 8)) = False Then
        count1 = count1 + 1
    Else
        If ToolOfflineData(i, 7) = 2 And IsEmpty(ToolOfflineData(i, 8)) = False Then
            count2 = count2 + 1
        End If
    End If
End If
Next i

ReDim rawPMdata(count1, 7)
ReDim rawDTdata(count2, 7)

count1 = 0
count2 = 0
For i = 1 To UBound(ToolOfflineData)
    If ToolOfflineData(i, 7) = 1 And IsEmpty(ToolOfflineData(i, 8)) = False Then
        count1 = count1 + 1
        rawPMdata(count1, 7) = ToolOfflineData(i, 8)
        For j = 1 To 6
            rawPMdata(count1, j) = ToolOfflineData(i, j)
        Next j
    Else
        If ToolOfflineData(i, 7) = 2 And IsEmpty(ToolOfflineData(i, 8)) = False Then
            count2 = count2 + 1
            rawDTdata(count2, 7) = ToolOfflineData(i, 8)
            For j = 1 To 6
                rawDTdata(count2, j) = ToolOfflineData(i, j)
            Next j
        End If
    End If
End If
Next i
End Sub

```

```

Sub GetDT()
    Dim y, i, counter As Integer
    Dim toolName(), downtimeTemp(), upTimeTemp() As Variant
    Dim OLID, startCell, endcell, startEvent, endEvent, nextEvent As Integer

    toolName = Tools

    ReDim DToutput(UBound(toolName), 3)
    For t = 1 To UBound(toolName)
        j = 1
        Do Until rawDTdata(j, 1) = toolName(t)
            j = j + 1
        Loop
        startCell = j

        k = 0
        Do Until rawDTdata(startCell + k, 1) <> toolName(t) Or startCell + k = UBound(rawDTdata)
            k = k + 1
        Loop

        If startCell + k = UBound(rawDTdata) Then 'the last in rawDTdata
            endcell = startCell + k
        Else
            endcell = startCell + k - 1
        End If

        counter = 0
        i = startCell
    Next t
End Sub

```

```

Do While i <= endcell
  OLID = rawDTdata(i, 7)
  startEvent = i
  Do Until rawDTdata(i, 7) <> OLID Or i >= endcell
    i = i + 1
  Loop

  endEvent = i - 1
  nextEvent = i

  If i < endcell Then
    counter = counter + 1
    ReDim Preserve downtimeTemp(counter)
    ReDim Preserve upTimeTemp(counter)
    downtimeTemp(counter) = (rawDTdata(endEvent, 2) - rawDTdata(startEvent, 2)) * 24
    upTimeTemp(counter) = (rawDTdata(nextEvent, 2) - rawDTdata(endEvent, 2)) * 24
  Else
    i = i + 1
  End If
Loop

DToutput(t, 1) = t 'Tool Number not name
DToutput(t, 2) = WorksheetFunction.Average(upTimeTemp) 'Mean Uptime
DToutput(t, 3) = WorksheetFunction.Average(downtimeTemp) 'Mean Downtime
Next t

End Sub

```

```

Sub GetPM()
  Dim y, i, counter As Integer
  Dim toolName(), downtimeTemp(), upTimeTemp() As Variant
  Dim OLID, startCell, endcell, startEvent, endEvent, nextEvent As Integer

  toolName = Tools

  ReDim PMoutput(UBound(toolName), 3)
  For t = 1 To UBound(toolName)
    j = 1
    Do Until rawPMdata(j, 1) = toolName(t)
      j = j + 1
    Loop
    startCell = j

    k = 0
    Do Until rawPMdata(startCell + k, 1) <> toolName(t) Or startCell + k = UBound(rawPMdata)
      k = k + 1
    Loop

    If startCell + k = UBound(rawPMdata) Then 'the last in rawPMdata
      endcell = startCell + k
    Else
      endcell = startCell + k - 1
    End If

    counter = 0
    i = startCell
    Do While i <= endcell
      OLID = rawPMdata(i, 7)
      startEvent = i
      Do Until rawPMdata(i, 7) <> OLID Or i >= endcell
        i = i + 1
      Loop

      endEvent = i - 1
      nextEvent = i

      If i < endcell Then
        counter = counter + 1
        ReDim Preserve downtimeTemp(counter)
        ReDim Preserve upTimeTemp(counter)
        downtimeTemp(counter) = (rawPMdata(endEvent, 2) - rawPMdata(startEvent, 2)) * 24
        upTimeTemp(counter) = (rawPMdata(nextEvent, 2) - rawPMdata(endEvent, 2)) * 24
      Else
        i = i + 1
      End If
    Loop

    PMoutput(t, 1) = t 'Tool Number not name
    PMoutput(t, 2) = WorksheetFunction.Average(upTimeTemp) 'Mean Uptime
  Next t
End Sub

```

```

PMoutput(t, 3) = WorksheetFunction.Average(downtimeTemp) 'Mean Downtime
Next t
End Sub

```

Downtime userform

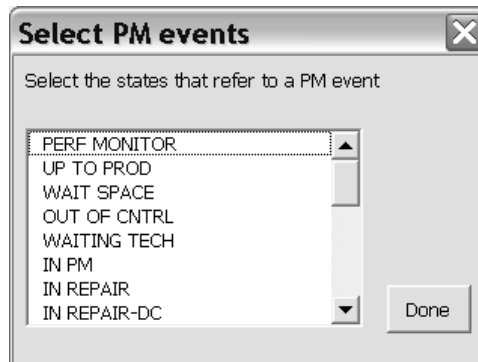


Figure B.1: User prompt to distinguish between scheduled and unscheduled downtime events recorded in the tool history.

```

Private Sub doneButton_Click()

Dim lItem, counter As Integer

counter = 0
For litem = 0 To DTUserform.ListBox1.ListCount - 1
If DTUserform.ListBox1.Selected(lItem) = True Then
counter = counter + 1
ReDim Preserve PMState(counter)
PMState(counter) = DTUserform.ListBox1.List(lItem)
DTUserform.ListBox1.Selected(lItem) = False
End If
Next lItem

Unload Me
End Sub

```

B.2.5 Lot selection parameters

This section includes the VB Userform code required for the selection and ranking of operation types for each tool. A full description is given in Section 4.3.4

ToolRank userform

Figure B.2: VB Userform used to select lot prioritisation options for each tool.

```

Private Sub UserForm_Initialize()

    Dim newlabel As MSForms.Label
    Dim newComboBox As MSForms.ComboBox
    Dim tempStr As String
    Dim x As Integer
    Dim recHeight As Double
    Dim actHeight As Double
    Dim dialogHeight As Double

    Application.VBE.MainWindow.Visible = False

    dialogHeight = 18
    y = UBound(Tools)

    'set numbers
    For x = 1 To y
        Set newlabel = ToolRank.Frame1.Controls.Add("Forms.label.1")
        With newlabel
            .Name = "RkToolNo" & x
            .Caption = x
            .Top = 10 + (dialogHeight * (x - 1))
            .Left = 10
            .Width = 40
            .Height = dialogHeight
            .Font.Size = 10
            .Font.Name = "Tahoma"
        End With
    Next x

    'set text
    For x = 1 To y
        Set newlabel = ToolRank.Frame1.Controls.Add("Forms.label.1")
        With newlabel
            .Name = "RankToolName" & x
            .Caption = ": " & Tools(x)
            .Top = 10 + (dialogHeight * (x - 1))
            .Left = 24
            .Width = 70
            .Height = dialogHeight
            .Font.Size = 10
            .Font.Name = "Tahoma"
        End With
    Next x

    'set comboboxes
    For x = 1 To y

```

```

Set newComboBox = ToolRank.Frame1.Controls.Add("Forms.combobox.1")
With newComboBox
    .Name = x
    .AddItem ("1. No ranking of operations")
    .AddItem ("2. Rank operations by last run process")
    .AddItem ("3. Rank operations by historical data")
    .AddItem ("4. Manually input operation ranks")
    .Top = 10 + (dialogHeight * (x - 1))
    .Left = 100
    .Width = 180
    .Height = dialogHeight
    .Font.Size = 10
    .Font.Name = "Tahoma"
    .EnterFieldBehavior = fmEnterFieldBehaviorRecallSelection
    .MatchEntry = fmMatchEntryFirstLetter
    .MatchRequired = True
    .ListIndex = 0
End With
Next x

actHeight = 240
recHeight = 20 + (dialogHeight * x)
If recHeight > actHeight Then
    Me.Frame1.ScrollBars = fmScrollBarsVertical
    Me.Frame1.ScrollHeight = recHeight
End If

Application.VBE.MainWindow.Visible = True

End Sub

```

```

Private Sub RankOKButton_Click()

    Dim ctrl As Control
    Dim selIdentifier, count, i, j, k As Integer
    Dim tempInfo(), tempOpID() As Variant
    Dim toolName As String
    Dim Temp As Variant
    Dim changedFlag As Boolean
    Dim count1 As Integer

    count1 = 0
    For Each ctrl In Me.Frame1.Controls
        If TypeName(ctrl) = "ComboBox" Then
            selIdentifier = Mid(ctrl.value, 1, 1)

            count = 0
            For i = 1 To UBound(pTimes)
                If ctrl.Name = pTimes(i, 1) Then
                    count = count + 1
                    ReDim Preserve tempOpID(count)
                    tempOpID(count) = pTimes(i, 3)
                End If
            Next i

            ReDim tempInfo(count, 3)
            For j = 1 To count
                tempInfo(j, 1) = tempOpID(j)
            Next j

            Select Case selIdentifier
                Case 1 'no ranking
                    For z = 1 To count
                        tempInfo(z, 2) = 0
                        tempInfo(z, 3) = tempInfo(z, 1)
                    Next z

                Case 2 'last processed ranking
                    For z = 1 To count
                        tempInfo(z, 2) = -1
                        tempInfo(z, 3) = tempInfo(z, 1)
                    Next z

                Case 3 'product ratio priority detected by FlexiSim
                    For z = 1 To count
                        k = 1
                        Do Until pTimes(k, 1) = ctrl.Name And pTimes(k, 3) = tempInfo(z, 1)
                            k = k + 1
                        Loop

```

```

    tempInfo(z, 3) = pTimes(k, 10)
Next z

'Rank them by the highest in column 3
For i = 1 To count - 1
    For j = i + 1 To count
        If tempInfo(j, 3) > tempInfo(i, 3) Then
            Temp = tempInfo(i, 3)
            tempInfo(i, 3) = tempInfo(j, 3)
            tempInfo(j, 3) = Temp
            Temp = tempInfo(i, 1)
            tempInfo(i, 1) = tempInfo(j, 1)
            tempInfo(j, 1) = Temp
        End If
    Next j
Next i

For m = 1 To count
    tempInfo(m, 2) = m
    tempInfo(m, 3) = tempInfo(m, 1)
Next m

Case 4 'user selects priorities
count1 = count1 + 1
ReDim Preserve passedToolID(count1)
passedToolID(count1) = ctrl.Name

Case Else
MsgBox ("Warning error in selecting operation ranks in UserForm ToolRank. Operation
        aborted")
Exit Sub
End Select

If Not (selIdentifier = 4) Then
    toolName = "gaOpRank" & ctrl.Name
End If
End If
Next ctrl

Unload Me

If count1 > 0 Then
    GetManualOpRanks
End If

End Sub

```

```

Private Sub RankCancelButton_Click()
    Unload Me
End Sub

```

RankOps userform

```

Private Sub UserForm_Initialize()

    Dim toolID, x As Integer
    Dim toolName As String
    Dim tempOpID() As Variant
    Dim newlabel As MSForms.Label
    Dim newComboBox As MSForms.ComboBox
    Dim dialogHeight As Double
    Dim count As Integer

    toolName = CStr(Tools(CInt(passedToolID(toolIdentifier))))
    ToolIDLabel.Caption = toolName

    count = 0
    For i = 1 To UBound(pTimes)
        If passedToolID(toolIdentifier) = pTimes(i, 1) Then
            count = count + 1
            ReDim Preserve tempOpID(count)
        End If
    Next i

```

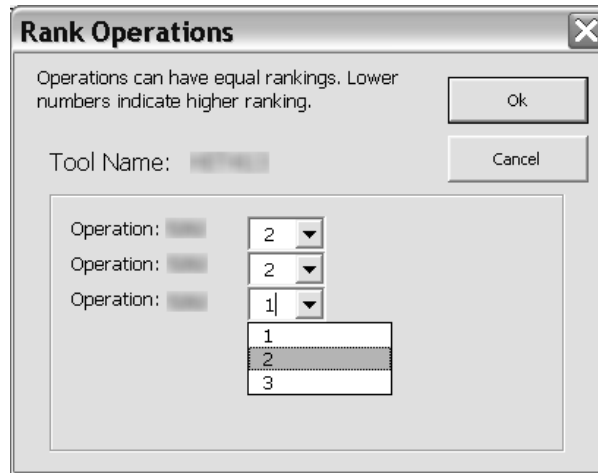


Figure B.3: VB Userform used to rank processing priority for operations.

```

        tempOpID(count) = pTimes(i, 3)
    End If
Next i

dialogHeight = 18

'set numbers
For x = 1 To UBound(tempOpID)
    Set newlabel = RankOps.Frame1.Controls.Add("Forms.Label.1")
    With newlabel
        .Name = "OpLabel" & x
        .Caption = "Operation: " & tempOpID(x)
        .Top = 10 + (dialogHeight * (x - 1))
        .Left = 10
        .Width = 80
        .Height = dialogHeight
        .Font.Size = 10
        .Font.Name = "Tahoma"
    End With
Next x

'set comboboxes
For x = 1 To UBound(tempOpID)
    Set newComboBox = RankOps.Frame1.Controls.Add("Forms.combobox.1")
    With newComboBox
        .Name = "ComboBox" & x
        For j = 1 To UBound(tempOpID)
            .AddItem (j)
        Next j
        .Top = 10 + (dialogHeight * (x - 1))
        .Left = 100
        .Width = 40
        .Height = dialogHeight
        .Font.Size = 10
        .Font.Name = "Tahoma"
        .EnterFieldBehavior = fmEnterFieldBehaviorRecallSelection
        .MatchEntry = fmMatchEntryFirstLetter
        .MatchRequired = True
        .ListIndex = 0
    End With
Next x

actHeight = 132
recHeight = 20 + (dialogHeight * x)
If recHeight > actHeight Then
    Me.Frame1.ScrollBars = fmScrollBarsVertical
    Me.Frame1.ScrollHeight = recHeight
End If
End Sub

```

```
Private Sub OkButton_Click()
```

```

Dim ctrl As Control
Dim tempArray1(), tempArray2(), finalArray() As Variant
Dim tempStr, toolName As String
Dim j As Integer

count = 0
count2 = 0
For Each ctrl In Me.Frame1.Controls
    If TypeName(ctrl) = "Label" Then
        count = count + 1
        ReDim Preserve tempArray1(count)
        tempArray1(count) = Mid(ctrl, 12, 4)
    End If

    If TypeName(ctrl) = "ComboBox" Then
        count2 = count2 + 1
        ReDim Preserve tempArray2(count2)
        tempArray2(count2) = ctrl.value
    End If
Next ctrl

ReDim finalArray(count, 3)
For i = 1 To UBound(finalArray)
    finalArray(i, 1) = tempArray1(i)
    finalArray(i, 2) = tempArray2(i)
    finalArray(i, 3) = finalArray(i, 1)
Next i

'Arrange in ranking order by column 2
For i = 1 To UBound(finalArray) - 1
    For j = i + 1 To UBound(finalArray)
        If finalArray(j, 2) < finalArray(i, 2) Then
            Temp = finalArray(i, 2)
            finalArray(i, 2) = finalArray(j, 2)
            finalArray(j, 2) = Temp
            Temp = finalArray(i, 1)
            finalArray(i, 1) = finalArray(j, 1)
            finalArray(j, 1) = Temp
        End If
    Next j
Next i

'Reset col3 = col1
For i = 1 To UBound(finalArray)
    finalArray(i, 1) = finalArray(i, 3)
Next i

toolName = "gaOpRank" & passedToolID(toolIdentifier)
If Not (PassArraytoExtendSim(finalArray, toolName)) Then
    MsgBox ("Warning. error when passing operation rank data to ExtendSim in Userform ToolRank
    . Operation aborted.")
    Exit Sub
End If

Unload Me
End Sub

```

```

Private Sub CancelButton_Click()
    Unload Me
End Sub

```


B.3 VBA Wrapper for ExtendSim Commands and Functions

This section includes some custom functions written in Visual Basic for the FTM Application to control and interact with the ExtendSim model. The functions rely heavily on *execute*, *request* and *poke* methods described by the ExtendSim manual (ExtendSim, 2009). The procedure and functions include:

- RetrieveModel Opens the ExtendSim model file.
- GetExtendAppPath Finds the local ExtendSim installation.
- PassAllDataToExtendSim Sends all the data and distribution information from VB to ExtendSim.
- RunExtendSimModel Runs the model.
- SaveAndCloseExtendSimModel Saves the model and cleanly exits ExtendSim.
- PassArrayToExtendSim Creates an ExtendSim array and populates it with the contents of a VB array.
- PassRunInfo Informs ExtendSim of some essential run parameters such as start time, end time and the number of replication to perform.
- ExtendDBTableWrite Writes an ExtendSim database to a text file in the root folder of the ExtendSim application.
- ReceiveDBfromExtendSim Creates an array in VB and populates it with an ExtendSim array.
- ReceiveArrayfromExtendSim Converts a VB array to string and passes it to a newly created ExtendSim array.
- SimAnimation Turns the simulation animation off.

```
Public Function RetrieveModel(moxfile As String)
    Dim GettingObject As Integer
    modelFile = moxfile
    On Error GoTo ErrorHandler:
```

Appendix B. FTM Application Code

```
GettingObject = 1

Set ExtendApp = GetObject(, "Extend.Application")

loadmodel:
GettingObject = 0
ExtendApp.Execute "OpenExtendFile(" + """" + modelFile + """" + ");"
GoTo Done:

ErrorHandler:
If GettingObject Then
    ' ExtendSim is not running (otherwise GetObject would have worked) so we
    ' call create object to start ExtendSim
    Set ExtendApp = CreateObject("Extend.Application")
    GoTo loadmodel:
Else
    MsgBox Error$
    Set ExtendApp = Nothing
End If

Done:

End Function
```

```
Public Sub GetExtendAppPath()
    ExtendApp.Execute "globalStr2 =getAppPath();"
    ExtendAppPath = ExtendApp.Request("System", "globalStr2+:0:0:0")
End Sub
```

```
Sub PassAllDataToExtendSim()

    Dim extendArray As String

Arrival:
    extendArray = "gaArrivalInfo"
    If PassArraytoExtendSim(arrivalInfo, extendArray) = False Then
        MsgBox "Error ocured when passing to " + extendArray + " . in sub DoProgram."
    End If

Tools:
    extendArray = "gaTool"
    If PassArraytoExtendSim(pTimes, extendArray) = False Then
        MsgBox "Error ocured when passing to " + extendArray + " . in sub DoProgram."
    End If

Downtime:
    extendArray = "gaDT"
    If PassArraytoExtendSim(DToutput, extendArray) = False Then
        MsgBox "Error ocured when passing to " + extendArray + " . in sub DoProgram."
    End If

PM:
    extendArray = "gaPM"
    If PassArraytoExtendSim(PMoutput, extendArray) = False Then
        MsgBox "Error ocured when passing to " + extendArray + " . in sub DoProgram."
    End If

End Sub
```

```
Sub RunExtendSimModel()
    ExtendApp.Execute "ExecuteMenuCommand(6000);" 'Use the ExecuteMenuCommand() function to run
    the simulation
End Sub
```

```
Sub SaveAndCloseExtendSimModel()
    ExtendApp.Execute "ExecuteMenuCommand(5);" 'close the model
```

Appendix B. FTM Application Code

```

ExtendApp.Execute "ExecuteMenuCommand(4);" 'Use the ExecuteMenuCommand() function to save
the simulation
ExtendApp.Execute "ExecuteMenuCommand(1);" 'close Extend
Set ExtendApp = Nothing 'Destroy the extendApp
End Sub

```

```

Public Sub PassArraytoExtendSim(dataArray() As Variant, extendArray As String)

    'Converts the dataArray to string and passes it to a newly created extendArray
    Dim i, j, x, y As Integer
    Dim strArray() As String
    Dim n, m As String
    Dim blockNum() As Integer

    On Error GoTo ErrorHandler:

    y = UBound(dataArray, 1)
    x = UBound(dataArray, 2)
    ReDim strArray(y, x)

    For i = 1 To y
        For j = 1 To x
            strArray(i, j) = CStr(dataArray(i, j))
        Next j
    Next i

    ExtendApp.Execute "globalStr2 =GAGetIndex(" + """" + extendArray + """" + ");"
    globalIndex = ExtendApp.Request("System", "globalStr2+:0:0:0")

    If globalIndex > -1 Then 'delete array and recreate it
        ExtendApp.Execute "globalStr2 =GaDispose(" + """" + extendArray + """" + ");"
    End If

    ExtendApp.Execute "globalStr2 =GACreate(" + """" + extendArray + """" + ",1," & x & ");"
    globalIndex = ExtendApp.Request("System", "globalStr2+:0:0:0")

    If globalIndex < 0 Then
        MsgBox "Failed to create the global array named " + extendArray + " in " + modelFile
        PasstoExtendSim = False
        Exit Sub
    End If

    globalIndex = val(globalIndex)
    ExtendApp.Execute "GAResize(" + """" + extendArray + """" + ", " & y & ");"

    For i = 1 To y
        For j = 1 To x
            m = CStr(i - 1)
            n = CStr(j - 1)
            ExtendApp.Execute "GASetReal(" & strArray(i, j) & "," & globalIndex & "," + m + "," + n
            + ");"
        Next j
    Next i

    ErrorHandler:
    If Err.Number <> 0 Then
        MsgBox Error$ & "Error occured when passing to array " + extendArray + " to simulation.
        Operation aborted."
        Exit Sub
    End If

End Sub

```

```

Public Sub PassRunInfo(value As Variant, infoType As Integer)

    'This function passes information to ExtendSim that refers to its sim setup variable the
    permitted infoTypes are
    SimEndTime = 1
    SimStartTime = 2
    SimReps = 3

    Dim val, which As String
    Dim returnValue As Integer

    If infoType > 3 Or infoType < 1 Then

```

Appendix B. FTM Application Code

```

MsgBox ("Undefined information type in function PassRunInfo. Operation aborted.")
Exit Sub
End If

val = CStr(value)
which = CStr(infoType)

ExtendApp.Execute "globalStr2 = SetRunParameter(" & val & "," & which & ");"
returnValue = ExtendApp.Request("System", "globalStr2+:0:0:0")

If returnValue <> 1 Then
    MsgBox ("Error returned form ExtendSim in function PassRunInfo. Operation aborted.")
    Exit Sub
End If

End Sub

```

```

Public Sub ExtendDBTableWrite(dbName As String, tableName As String, textFileName As String)

    'Writes a database to a text file in the root folder of the ExtendSim application
    Dim dbNameIndex As Integer
    Dim tableNameIndex As Integer
    Dim numofRecords, numofFields As Long
    Dim strDelim, strPrompt As String
    Dim successFlag As Long

    'Get DB Index
    ExtendApp.Execute "globalStr2 = DBDatabaseGetIndex(" + """" + dbName + """" + ");"
    dbNameIndex = ExtendApp.Request("System", "globalStr2+:0:0:0")

    If dbNameIndex <= 0 Then
        MsgBox ("Warning database " & dbName & " not found in ExtendSim. Operation aborted")
        Exit Sub
    End If

    'Get Table Index
    ExtendApp.Execute "globalStr2 = DBTableGetIndex(" & dbNameIndex & "," + """" + tableName + """" + """" + ");"
    tableNameIndex = ExtendApp.Request("System", "globalStr2+:0:0:0")

    If tableNameIndex <= 0 Then
        MsgBox ("Warning table " & tableNameIndex & " not found in ExtendSim. Operation aborted")
        Exit Sub
    End If

    'Get number of Records
    ExtendApp.Execute "globalStr2 = DBRecordsGetNum(" & dbNameIndex & "," & tableNameIndex & ");"
    numofRecords = ExtendApp.Request("System", "globalStr2+:0:0:0")

    'Get number of fields
    ExtendApp.Execute "globalStr2 = DBFieldsGetNum(" & dbNameIndex & "," & tableNameIndex & ");"
    numofFields = ExtendApp.Request("System", "globalStr2+:0:0:0")

    strDelim = ","
    strPrompt = ""

    ExtendApp.Execute "globalStr2 = DBTableExportData(" + """" + textFileName + """" + "," + """" + """" + strPrompt + """" + "," + """" + strDelim + """" + "," + """" & dbNameIndex & """" + """" & tableNameIndex & """" + """" & numofRecords & """" + """" & numofFields & """);"
    successFlag = val(ExtendApp.Request("System", "globalStr2+:0:0:0"))

    If successFlag = -1 Then
        MsgBox ("Error occurred during data export from simulation. Operation aborted.")
        Exit Sub
    End If

End Sub

```

```

Public Function RecieveDBfromExtendsim(dbName As String, tableName As String) As Variant

    Dim dbNameIndex As Integer
    Dim tableNameIndex As Integer
    Dim numofRecords, numofFields As Integer
    Dim dataArray() As Variant

```

Appendix B. FTM Application Code

```

Dim i, j, lastusedRow As Integer

'Get DB Index
ExtendApp.Execute "globalStr2 = DBDatabaseGetIndex(" + """" + dbName + """" + ");"
dbNameIndex = ExtendApp.Request("System", "globalStr2+:0:0:0")

If dbNameIndex <= 0 Then
    MsgBox ("Warning database " & dbName & " not found in ExtendSim. Operation aborted")
    Exit Function
End If

'Get Table Index
ExtendApp.Execute "globalStr2 = DBTableGetIndex(" & dbNameIndex & "," + """" + tableName + """" + ");"
tableNameIndex = ExtendApp.Request("System", "globalStr2+:0:0:0")

If tableNameIndex <= 0 Then
    MsgBox ("Warning table " & tableNameIndex & " not found in ExtendSim. Operation aborted")
    Exit Function
End If

'Get number of Records
ExtendApp.Execute "globalStr2 = DBRecordsGetNum(" & dbNameIndex & "," & tableNameIndex & ");"
numOfRecords = ExtendApp.Request("System", "globalStr2+:0:0:0")

'Get number of fields
ExtendApp.Execute "globalStr2 = DBFieldsGetNum(" & dbNameIndex & "," & tableNameIndex & ");"
numOfFields = ExtendApp.Request("System", "globalStr2+:0:0:0")

ReDim dataArray(numOfRecords, numOfFields)

For i = 1 To numOfRecords
    For j = 1 To numOfFields
        ExtendApp.Execute "globalStr2 = DBDataGetAsNumber(" & dbNameIndex & "," & tableNameIndex & "," & j & "," & i & ");"
        dataArray(i, j) = val(ExtendApp.Request("System", "globalStr2+:0:0:0"))
    Next j
Next i

RecieveDBfromExtendsim = dataArray

End Function

```

```

Public Function RecieveArrayfromExtendsim(extendArray As String) As Variant

'Converts the dataArray to string and passes it to a newly created extendArray
Dim i, j, globalIndex, NumRows, numCols As Integer
Dim dataArray() As Variant

ExtendApp.Execute "globalStr2 =GAGetIndex(" + """" + extendArray + """" + ");"
globalIndex = ExtendApp.Request("System", "globalStr2+:0:0:0")

ExtendApp.Execute "globalStr2 =GAGetRows(" + """" + extendArray + """" + ");"
NumRows = ExtendApp.Request("System", "globalStr2+:0:0:0")

ExtendApp.Execute "globalStr2 =GAGetColumns(" + """" + extendArray + """" + ");"
numCols = ExtendApp.Request("System", "globalStr2+:0:0:0")

ReDim dataArray(NumRows, numCols)

For i = 1 To NumRows
    For j = 1 To numCols
        m = CStr(i - 1)
        n = CStr(j - 1)
        ExtendApp.Execute "globalStr2 = GAGetReal(" & globalIndex & "," + m + "," + n + ");"
        dataArray(i, j) = ExtendApp.Request("System", "globalStr2+:0:0:0")
    Next j
Next i

RecieveArrayfromExtendsim = dataArray

End Function

```

```

Public Function SimAnimation(TRUEorFALSE As Boolean)

```

```

Dim IsAnimationOn As Boolean

ExtendApp.Execute "globalStr2 =AnimationOn;"
IsAnimationOn = ExtendApp.Request("System", "globalStr2+:0:0:0")

If Not (IsAnimationOn) And TRUEorFALSE Then 'Then Anim is OFF and we want it turned ON
  ExtendApp.Execute "ExecuteMenuCommand(2020);" 'change the animation from ON to OFF
ElseIf IsAnimationOn And Not (TRUEorFALSE) Then 'Then Anim is ON and we want it OFF
  ExtendApp.Execute "ExecuteMenuCommand(2020);" 'change the animation from ON to OFF
End If

End Function

```

B.4 ExtendSim Custom Blocks

This section includes the custom blocks coded in ModL for the ExtendSim model in the FTM application.

B.4.1 Lot generator code

The lot generator block reads the *ArrivalInfo* array which contains the necessary information about the lot arrival patterns. It then reconstructs the distributions, samples from them and creates and releases lots during model runtime.

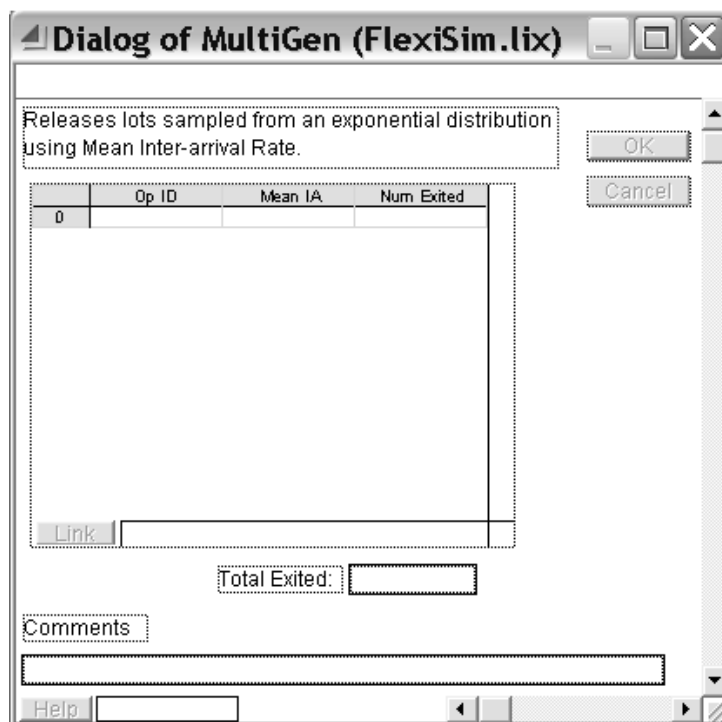


Figure B.4: Dialog of custom Lot Generator block for FTM Application.

```

real itemArrayR[][3];
integer itemArrayI[][5];
integer ItemArray3D[][10];
string itemArrayT[];
real itemArrayC[][10];
real timeArray[];
integer timeblocks[];
integer timeEventMsgType[];
integer exec;
integer rCount;
integer intArray[][2];
integer itemOutIsConnected;
integer numofAttribs;
integer opIDAttribExist;
integer attribListIndex;
integer itemIndex;
integer sending;
integer attribListCols;
integer OpIDColIndex;
integer itemTypeColIndex;
integer attribValueIndex;
integer arrivalInfoIndex;
integer gaArrivalInfoIndex;
real ArrivalInfo[][3];
integer nextRelease;
integer nextOpForRelease;
integer i,j;
integer myindex;
integer myNumber;
integer downstreamBlockNumber;

constant rejects is 0;
constant wants is 1;
constant taken is 2;
constant needs is 3;
constant query is 4;
constant notify is 5;
constant blocked is 6;
constant init is 7;

#include "CheckVersion v7.h";

//this procedure takes in all the itemArray information controlled by the executive
procedure getArrays()
{
  if (rCount != sysGlobalint2)
  {
    getPassedArray(sysGlobal3, itemArrayR);
    getPassedArray(sysGlobal4, itemArrayI);
    getPassedArray(sysGlobal9, itemArrayC);
    getPassedArray(sysGlobal12, itemArray3D);
    rCount = sysGlobalint2;
  }
}

//this function is called to create an item by interacting with executive
integer createItem()
{
  integer value;
  getArrays();

  if (sysGlobalint1+1 >= sysGlobalint2) //the first free row of the item index list
  {
    sendMsgToBlock(exec, blockreceive0Msg); // expand array if not big enough
    getArrays();
  }

  ItemIndex = sysGlobalint1; //the value is now the items index number
  itemArrayI[itemIndex][0] = 0; // row used
  itemArrayI[itemIndex][1] = 0; // batch ID
  itemArrayI[itemIndex][2] = 0; // user value
  itemArrayI[itemIndex][3] = 0; // batch ID #2
  itemArrayI[itemIndex][4] = 0; // unused
  itemArrayR[itemIndex][0] = 1.0; // value
  itemArrayR[itemIndex][1] = BLANK; // Priority
  itemArrayR[itemIndex][2] = BLANK; // user value

  //update the attribute value with the opID name
  GaSetReal (NextOpForRelease, attribValueIndex, itemIndex, OpIDColIndex);
  GaSetReal (1, attribValueIndex, itemIndex, itemTypeColIndex); //will always be 1
  GaSetReal (66, attribValueIndex, itemIndex, 0); //0 for animation, 66 for green circle

```

```

sendMsgToBlock(exec, blockReceiveMsg); // check for next space
return(itemIndex);
}

//This function was called from 'on blockReceive'
procedure SendItem()
{
    sending = TRUE;
    if(itemOut > 0.0) //if an item is ready to leave
    {
        sysGlobalInt3 = wants; //tell upstream block we are ready
        sendMsgToInputs(itemOut); //upstream will check if it can accept
    }

    if(sysGlobalInt0 == needs)
    {
        sysGlobalInt3 = needs;
        sendMsgToInputs(itemOut);

        //Any Statistics
        i=0;
        while (nextOpforRelease!=ArrivalInfo[i][0])
        {
            i++;
        }

        ArrivalInfo[i][2]=strToReal(ArrivalInfo[i][2])+1;
        RefreshDatatableCells(myNumber, "MultiGenTable", 0, 0, getDimension(MultiGenTable),
            getdimensioncolumns(MultiGenTable));
        TotalExited = TotalExited+1;
    }

    if(itemOut >= 0.0)
    {
        Usererror("Failure");
        abort;
    }

    sending = FALSE;
}

// If the dialog data is inconsistent for simulation, abort.
on checkdata
{
    exec = sysGlobalInt1; //block number for the executive block
    myIndex=sysGlobalInt0; //assigns a unique integer for positioning in TimeArray and
        TimeBlockArray
    sysGlobalInt0 += 1; //updates for the next block

    //Initialse the table to blanks
    for (i=0;i<GetDimension(MultiGenTable);i++)
    {
        for (j=0;j<GetDimensionColumns(MultiGenTable);j++)
        {
            MultiGenTable[i][j]=BLANK;
        }
    }

    //ensure that gaarrivalInfo exists
    gaArrivalInfoIndex = gagetindex("gaArrivalInfo");
    if (gaArrivalInfoIndex <0)
    {
        UserError("Warning global array gaArrivalInfo does not exist. Report from "+ myblocknumber
            ()+".");
        abort;
    }
}

// stepsize
on Stepsize
{
    //set the OpID Attribute.
    attribListIndex = GAGetIndex("_AttributeList");
    numofAttribs = GAGetRowsByIndex (attribListIndex) ;

    //check if it already exists
    if(GAFindStringAny(attribListIndex, "opID", 0, numofAttribs, 4, FALSE) <0)
    {
        GAResizeByIndex (AttribListIndex, numofAttribs+1);
        GaSetString15("opID", attribListIndex, numofAttribs, 0);
    }

    //check if itemType exists, if not add it to the attribute list

```



```

numofAttribs = GAGetRowsByIndex (attribListIndex) ;
if (GAFindStringAny (attribListIndex, "itemType", 0, numofAttribs, 8, FALSE) < 0)
{
    GAResizeByIndex (AttribListIndex, numofAttribs+1);
    GaSetString15 ("itemType", attribListIndex, numofAttribs, 0);
}

numofAttribs = GAGetRowsByIndex (attribListIndex);
}

// Initialize any simulation variables.
on initsim
{
    itemIndex = 0;
    rCount = -1; //row count for item index
    myNumber = MyBlockNumber();
    TotalExited = 0;

    //check to see if executive has been placed properly
    if (GetPassedArray (sysGlobal0, timeArray) )
    {
        CheckVersionExec();
        exec = sysGlobalInt23;
    }
    else
    {
        UserError ("The Executive block must be present and to the left of all blocks on the
            worksheet.");
        AbortAllSims();
    }

    // get the pointer to the TimeArray and TimeEventMsgType arrays
    if (getPassedArray (SysGlobal0, timeArray) > 0)
    {
        // set the first event time to the start of the simulation
        timeArray[MyIndex] = StartTime;
        // get the pointer to the TimeBlocks array
        getPassedArray (SysGlobal7, TimeBlocks);
        // put this block's # in reserved position in TimeBlocks
        TimeBlocks[myindex] = myBlockNumber();
        //Get the pointer to the TimeEventMsgType array
        getPassedArray (SysGlobal13, TimeEventMsgType);
        //reserved position in TimeEventMsgType
        TimeEventMsgType[myIndex] = BlockReceiveMsg;
    }

    GetSimulateMsgs (FALSE); //this is discrete event block. No simulate messages

    //find the output connector block number and store as downstreamNumber
    if (novalue (GetConnectedTextBlock (myNumber, getConNumber (myNumber, "ItemOut"))))
    {
        userError ("Output not connected in 'MultiGen' block number " + myNumber + ".Recorrect");
        abort;
    }

    //Ensure downstream block is a queue block.
    if (GetConnectedTextBlock (myNumber, getConNumber (myNumber, "ItemOut")) < 0)
    {
        GetConBlocks (myNumber, getConNumber (myNumber, "ItemOut"), intArray);
        If (getdimension (intArray) <= 0)
        {
            UserError ("Output not connected in 'MultiGen' block number " + myNumber + ".Recorrect");
            abort;
        }

        downstreamBlockNumber = intArray[0][0];
        itemOutIsConnected = TRUE;

        if (GetBlockType (downstreamBlockNumber) != "Queues")
        {
            UserError ("Item is blocked from leaving 'MultiGen' block number " + MyBlockNumber() + ".
                Place a queue after the 'Create' to prevent items from being destroyed.");
            itemOutIsConnected = FALSE;
            AbortAllSims();
        }
    }

    //Read in the values from the global array named
    //gaArrivalInfo into local dynamic array named ArrivalInfo
    makeArray (ArrivalInfo, GAGetrowsByIndex (gaArrivalInfoIndex));
    for (i=0; i < GAGetRowsByIndex (gaArrivalInfoIndex); i++)
    {
        ArrivalInfo[i][0] = GAGetReal (gaArrivalInfoIndex, i, 0);
    }
}

```

```

ArrivalInfo[i][1]=GAGetReal(gaArrivalInfoIndex,i,1);
ArrivalInfo[i][2]=0;
}

//set up table
DynamicDataTable(myNumber, "MultiGenTable", ArrivalInfo);
RefreshDatatableCells(myNumber, "MultiGenTable", 0, 0, getDimension(MultiGenTable),
    getdimensioncolumns(MultiGenTable));

//Find the row index of the attribute OpID in ("_AttributeList")
numofAttribs = GAGetRowsByIndex (attribListIndex) ;
if (GAFindStringAny (attribListIndex, "opID", 0, numofAttribs, 4, FALSE) >=0)
{
    OpIDColIndex = 1+ GAFindStringAny (attribListIndex, "opID", 0, numofAttribs, 4, FALSE);
}

//Find the row index of the attribute itemtype in ("_AttributeList")
numofAttribs = GAGetRowsByIndex (attribListIndex) ;
if (GAFindStringAny (attribListIndex, "itemType", 0, numofAttribs, 8, FALSE) >=0)
{
    itemTypeColIndex = 1+ GAFindStringAny (attribListIndex, "itemType", 0, numofAttribs, 8, FALSE);
}

attribValueIndex = GaGetIndex("_AttribValues");

//initialise list to store next operation release time
nextRelease =ListCreate(myNumber, 1, 1, 0, 0, 0, 1, 0);
ListCreateElement(myNumber, nextRelease);

for (i = 0; i < GetDimension(ArrivalInfo); i++)
{
    //setup list
    ListSetLong(myNumber, nextRelease,-1,0, ArrivalInfo[i][0]);
    ListSetDouble(myNumber, nextRelease,-1,0,0);
    ListAddElement(myNumber, nextRelease, 2);
}

ListSetSort(myNumber, nextRelease, 1, 0) ; //resort the list

//get NextOpForRelease post to exec
NextOpForRelease = ListGetLong(myNumber, nextRelease, 0, 0);
TimeArray[myIndex] = ListGetDouble(myNumber, nextRelease, 0, 0);
}

//we receive this message from the executive because we posted and event
on BlockReceivel
{
    // create an item and set the output to its index value
    ItemOut = CreateItem();
    // initiate the send item prodedure
    SendItem();
    //Update nextReleaseTime and post event to executive
    i=0;
    while (ArrivalInfo[i][0]!=NextOpForRelease)
    {
        i++;
    }

    ListDeleteElement(myNumber, nextRelease, 0);
    ListSetLong(myNumber, nextRelease,-1,0, ArrivalInfo[i][0]);
    ListSetDouble(myNumber, nextRelease,-1,0,currenttime+DExponential(1/(ArrivalInfo[i][1]]));
    ListAddElement(myNumber, nextRelease, 2);
    ListSetSort(myNumber, nextRelease, 1, 0) ; //resort the list
    NextOpForRelease = ListGetLong(myNumber, nextRelease, 0, 0);
    TimeArray[myIndex] = ListGetDouble(myNumber, nextRelease, 0, 0);
}

on endsim
{
    ListDispose(myNumber, nextRelease);
}

```

B.4.2 Tool generator code

The tool generator block reads the *gaTool* global array that was populated by the VB front end. At simulation time zero, the block creates a tool *item* for each tool in the array and attaches the attribute information such as tool name, tool type, allowable operations, processing distribution information for each operation on that tool (Johnson parameters) and the mean move-out time for that tool. The tool *items* are then released into the model.

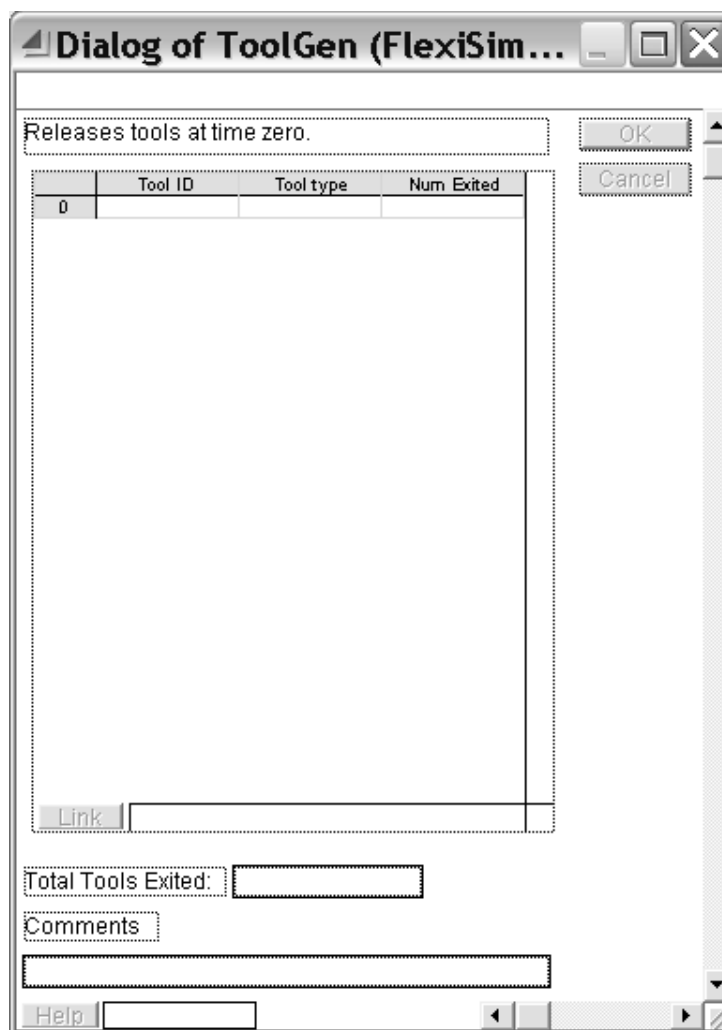


Figure B.5: Dialog of custom Tool Generator block for FTM Application.

```
// Declare constants and static variables here.
real itemArrayR[][3];
integer itemArrayI[][5];
integer ItemArray3D[][10];
string itemArrayT[];
```

```

real itemArrayC[][10];
real timeArray[];
integer timeblocks[];
integer timeEventMsgType[];
integer exec;
integer rCount;           //number of rows in the itemArrays
integer intArray[][2];   //the connected outputs
integer itemOutIsConnected;
integer numofAttribs;
integer attribListIndex;
integer itemIndex;
integer sending;
integer attribListCols;
integer attribValueIndex;
integer i,j;
integer myindex;
integer myNumber;
integer downstreamBlockNumber;
integer gaToolIndex;
integer toolInfoIndex;
integer numofTools;
integer numDeleted;
integer toolIDColumn;
integer toolTypeColumn;
integer itemTypeColumn;
integer toolForRelease;
integer toolTypeForRelease;
real ToolInfo[][3];

// item messages
constant rejects is 0;
constant wants   is 1;
constant taken   is 2;
constant needs   is 3;
constant query   is 4;
constant notify  is 5;
constant blocked is 6;
constant init    is 7;

//Include files:
#include "CheckVersion v7.h";

//-----
//this procedure takes in all the itemArray information controlled by the executive
procedure getArrays()
{
  if (rCount != sysGlobalint2)
  {
    getPassedArray(sysGlobal3, itemArrayR);
    getPassedArray(sysGlobal4, itemArrayI);
    getPassedArray(sysGlobal9, itemArrayC);
    getPassedArray(sysGlobal12, itemArray3D);
    rCount = sysGlobalint2;
  }
}

//this function is called to create an item by interacting with executive
integer createItem()
{
  integer value;

  getArrays();
  if (sysGlobalint1+1 >= sysGlobalint2) //the first free row of the item index list
  {
    sendMsgToBlock(exec, blockreceive0Msg); // expand array if not big enough
    getArrays();
  }

  ItemIndex = sysGlobalint1; //the value is now the items index number
  value = sysGlobalint1; //the value is now the items index number

  itemArrayI[value][0] = 0; // row used
  itemArrayI[value][1] = 0; // batch ID
  itemArrayI[value][2] = 0; // user value
  itemArrayI[value][3] = 0; // batch ID #2
  itemArrayI[value][4] = 0; // unused (not sure about this says, manual says its block
    number where item is
  itemArrayR[value][0] = 1.0; // value
  itemArrayR[value][1] = BLANK; // Priority
  itemArrayR[value][2] = BLANK; // user value

  //update the attribute value
  GaSetReal(toolForRelease,attribValueIndex,value,toolIDcolumn);

```

Appendix B. FTM Application Code

```

GaSetReal(toolTypeForRelease,attribValueIndex,value,tooltypecolumn);
GaSetReal(2,attribValueIndex,value,itemtypeColumn); //itemType is 2 because it is a tool.
GaSetReal(66,attribValueIndex,value,0); //0 for animation, 66 for green circle

sendMsgToBlock(exec, blockReceiveMsg); // check for next space
return(value);
}

//This function was called from 'on blockReceive'
procedure SendItem()
{
    sending = TRUE;
    if(itemOut > 0.0) //if an item is ready to leave
    {
        sysGlobalInt3 = wants; //tell upstreamblock we are ready
        sendMsgToInputs(itemOut); //upstream will check if it can accept
    }

    if(sysGlobalInt0 == needs)
    {
        sysGlobalInt3 = needs;
        sendMsgToInputs(itemOut);

        //Any Statistics
        ToolInfo[i][2]= ToolInfo[i][2]+1;
        TotalExitNumber = TotalExitNumber+1;
        RefreshDatatableCells(myNumber, "ToolTable", 0, 0, getDimension(ToolTable),
            getdimensioncolumns(ToolTable));
    }

    if(itemOut >= 0.0)
    {
        Usererror("Failure");
        abort;
    }

    sending = FALSE;
}

// If the dialog data is inconsistent for simulation, abort.
on checkdata
{
    exec = sysGlobalInt1; //block number for the executive block
    myIndex=sysGlobalInt0; //assigns a unique integer for positioning in TimeArray and
        TimeBlockArray
    sysGlobalInt0 += 1; //updates for the next block

    //Ensure that the global array gaTool exists
    GaToolIndex = GaGetIndex("gaTool");
    if (GaToolIndex<0)
    {
        Usererror("The global array gaTool does not exist. This simulation must be run from VB");
        abort;
    }

    DynamicDataTable(MyBlockNumber(), "ToolTable", ToolInfo);
}

on StepSize
{
    attribListIndex = GAGetIndex("_AttributeList");

    //check if toolID exists, if not add it to the attribute list
    numofAttribs = GAGetRowsByIndex (attribListIndex) ;
    if(GAFindStringAny(attribListIndex,"toolID",0,numofAttribs,6,FALSE) <0)
    {
        GAResizeByIndex (AttribListIndex,numofAttribs+1);
        GaSetString15("toolID",attribListIndex, numofAttribs, 0);
    }

    //check if toolType exists, if not add it to the attribute list
    numofAttribs = GAGetRowsByIndex (attribListIndex) ;
    if(GAFindStringAny(attribListIndex,"toolType",0,numofAttribs,8,FALSE) <0)
    {
        GAResizeByIndex (AttribListIndex,numofAttribs+1);
        GaSetString15("toolType",attribListIndex, numofAttribs, 0);
    }

    //check if itemType exists, if not add it to the attribute list
    numofAttribs = GAGetRowsByIndex (attribListIndex) ;
    if(GAFindStringAny(attribListIndex,"itemType",0,numofAttribs,8,FALSE) <0)
    {
        GAResizeByIndex (AttribListIndex,numofAttribs+1);
    }
}

```

```

    GaSetString15("itemType",attribListIndex, numofAttribs, 0);
}

numofAttribs = GAGetRowsByIndex (attribListIndex) ;
}

// Initialize any simulation variables.
on initsim
{
    itemIndex = 0;
    rCount = -1; //row count for item index
    myNumber = MyBlockNumber();
    TotalExitNumber = 0;

    //check to see if executive has been placed properly
    if( GetPassedArray(sysGlobal0, timeArray) )
    {
        CheckVersionExec(); //make sure the executive in the model is from the Item library
        exec = sysGlobalInt23;
    }
    else
    {
        UserError("The Executive block must be present and to the left of all blocks on the
            worksheet.");
        AbortAllSims();
    }

    // get the pointer to the TimeArray and TimeEventMsgType arrays
    if(getPassedArray(SysGlobal0, timeArray) > 0)
    {
        // set the first event time to the start of the simulation
        timeArray[MyIndex] = StartTime;
        // get the pointer to the TimeBlocks array
        getPassedArray(SysGlobal7,TimeBlocks);
        // put this block's # in reserved position in TimeBlocks
        TimeBlocks[myindex] = myBlockNumber();
        //Get the pointer to the TimeEventMsgType array
        getPassedArray(SysGlobal13,TimeEventMsgType);
        //reserved position in TimeEventMsgType
        TimeEventMsgType[myIndex] = BlockReceiveMsg;
    }

    attribValueIndex = GaGetIndex("_AttribValues");
    GetSimulateMsgs(FALSE); //this is discrete event block. No simulate messages

    //find the output connector block number and store as downstreamNumber
    if (novalue(GetConnectedTextBlock(myNumber, getConNumber(myNumber, "ItemOut"))))
    {
        userError ("Output not connected in block number " + myNumber + ".Recorrect");
        abort;
    }

    //Ensure downstream block is a queue block.
    if (GetConnectedTextBlock(myNumber, getConNumber(myNumber, "ItemOut"))<0)
    {
        GetConBlocks(myNumber, getConNumber(myNumber, "ItemOut"), intArray);
        If (getdimension(intArray)<=0)
        {
            UserError ("Output not connected in block number " + myNumber + ".Recorrect");
            abort;
        }

        downstreamBlockNumber = intArray[0][0];
        itemOutIsConnected = TRUE;

        if (GetBlockType(downstreamBlockNumber) != "Queues")
        {
            UserError ("Item is blocked from leaving block number " + MyBlockNumber() + ". Place a
                queue after it prevent items from being destroyed.");
            itemOutIsConnected = FALSE;
            AbortAllSims();
        }
    }

    //get the unique tools and read them into ToolInfo
    MakeArray(ToolInfo,gaGetRowsbyIndex(gaToolIndex)); //Expand the array

    For (i=0;i<getDimension(ToolInfo);i++)
    {
        ToolInfo[i][0]=gaGetReal(gaToolIndex,i,0);
        ToolInfo[i][1]=1; //for now delete this line and use next line when macro is set up
        //ToolInfo[i][1]=gaGetReal(gaToolIndex,i,1);
        ToolInfo[i][2]=0;
    }
}

```

```

    }

    //Sort the array and remove duplicate entries
    SortArray (ToolInfo, getDimension (ToolInfo), 0, TRUE, FALSE);

    i=getDimension (ToolInfo)-1;
    numDeleted = 0;
    while (i!=0)
    {
        If (ToolInfo[i][0]==ToolInfo[i-1][0])
        {
            ArrayDataMove (ToolInfo, i, 1, i-1, TRUE);
            numDeleted = numDeleted +1;
        }
        i=i-1;
    }

    numofTools = getDimension (ToolInfo)-numDeleted;
    SortArray (ToolInfo, getDimension (ToolInfo), 0, TRUE, FALSE);
    MakeArray (ToolInfo, numofTools);

    DynamicDataTable (myNumber, "ToolTable", ToolInfo);
    RefreshDatatableCells (myNumber, "ToolTable", 0, 0, getDimension (ToolTable),
        getdimensioncolumns (ToolTable));

    //Find the col location of ToolID, ToolType and ItemType
    numofAttribs = GAGetRowsByIndex (attribListIndex) ;
    if (GAFindStringAny (attribListIndex, "toolID", 0, numofAttribs, 6, FALSE) >=0)
    {
        toolIDColumn = 1+ GAFindStringAny (attribListIndex, "toolID", 0, numofAttribs, 6, FALSE);
    }

    if (GAFindStringAny (attribListIndex, "toolType", 0, numofAttribs, 8, FALSE) >=0)
    {
        toolTypeColumn = 1+ GAFindStringAny (attribListIndex, "toolType", 0, numofAttribs, 8, FALSE);
    }

    if (GAFindStringAny (attribListIndex, "itemType", 0, numofAttribs, 8, FALSE) >=0)
    {
        itemTypeColumn = 1+ GAFindStringAny (attribListIndex, "itemType", 0, numofAttribs, 8, FALSE);
    }

    TimeArray [myIndex] = startTime; //so that we get a block recieve1 message at time zero
}

//we receive this message from the executive because we posted and event
on BlockReceive1
{
    If (currentTime == StartTime)
    {
        i=0;
        For (i=0; i<numofTools;i++) //loop to ensure we get a BlockRecieve message until all
            tools are released
        {
            toolForRelease = ToolInfo [i][0];
            toolTypeForRelease = ToolInfo [i][1];
            ItemOut = CreateItem (); // create an item and set the output to its index value
            SendItem (); // initiate the send item prodedure
        }
    }

    //TimeArray [myIndex] = endtime; //ensure we get no more messages
}

```

B.4.3 Unscheduled downtime generator code

The unscheduled downtime generator block reads from the *gaDT* array the mean time before failure and the mean time to repair parameters of each tool. It then creates the downtime *items*, assigns them the MTBF and MTTR values as attributes and releases

them into the model at simulation time zero.

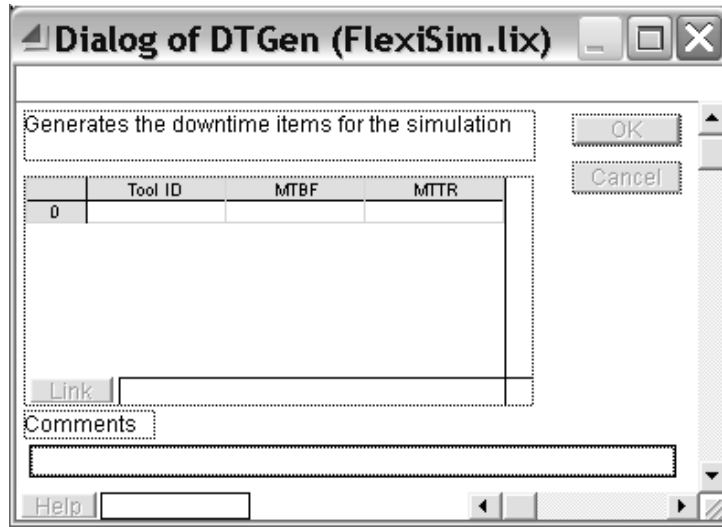


Figure B.6: Dialog of custom unscheduled downtime generator block for FTM Application.

```
// Constants
real   itemArrayR[][3];
integer itemArrayI[][5];
integer itemArray3D[][10];
string itemArrayT[];
real   itemArrayC[][10];
real   timeArray[];
integer timeblocks[];
integer timeEventMsgType[];
integer exec;
integer myIndex;
integer myNumber;
integer rCount;
integer intArray[][2];
integer downstreamBlockNumber;
integer itemOutisConnected;
integer i, j;
integer ItemIndex;
integer itemTypeCol;
integer toolTypeCol;
integer MTBFCol;
integer MTTRCol;
integer toolIDCol;
integer attribValueIndex;
integer attribListIndex;
integer numOfAttribs;
integer sending;
integer gaDTIndex;
integer dtInfoIndex;
real   DTInfo[][3];

// item messages
constant rejects      is 0;
constant wants       is 1;
constant taken       is 2;
constant needs       is 3;
constant query       is 4;
constant notify      is 5;
constant blocked     is 6;
constant init        is 7;

//this procedure takes in all the itemArray information controlled by the executive
procedure getArrays()
{
  if (rCount != sysGlobalint2)
  {
```



```

    getPassedArray(sysGlobal3, itemArrayR);
    getPassedArray(sysGlobal4, itemArrayI);
    getPassedArray(sysGlobal9, itemArrayC);
    getPassedArray(sysGlobal12, itemArray3D);
    rCount = sysGlobalint2;
  }
}

//this function is called to create an item by interacting with executive
integer createItem()
{
  getArrays();
  if (sysGlobalint1+1 >= sysGlobalint2) //the first free row of the item index list
  {
    sendMsgToBlock(exec, blockreceive0Msg); // expand array if not big enough
    getArrays();
  }

  ItemIndex = sysGlobalint1; //the value is now the items index number
  itemArrayI[ItemIndex][0] = 0; // row used
  itemArrayI[ItemIndex][1] = 0; // batch ID
  itemArrayI[ItemIndex][2] = 0; // user value
  itemArrayI[ItemIndex][3] = 0; // batch ID #2
  itemArrayI[ItemIndex][4] = 0; //
  itemArrayR[ItemIndex][0] = 1.0; // value
  itemArrayR[ItemIndex][1] = BLANK; // Priority
  itemArrayR[ItemIndex][2] = BLANK; // user value

  //update the attribute values.
  GaSetReal(dtInfo[i][0], attribValueIndex, itemIndex, toolIDCol);
  GaSetReal(dtInfo[i][1], attribValueIndex, itemIndex, MTBFCol);
  GaSetReal(dtInfo[i][2], attribValueIndex, itemIndex, MTTRCol);
  GaSetReal(3, attribValueIndex, ItemIndex, itemTypeCol); //itemType is 3 because it is a DT item

  GaSetReal(66, attribValueIndex, ItemIndex, 0); //0 for animation column, 65 for green circle

  sendMsgToBlock(exec, blockReceive1Msg); // check for next space
  return(ItemIndex);
}

//This function was called from 'on blockReceive1'
procedure SendItem()
{
  sending = TRUE;

  if(itemOut > 0.0) //if an item is ready to leave
  {
    sysGlobalInt3 = wants; //tell upstreamblock we are ready
    sendMsgToInputs(itemOut); //upstream will check if it can accept
  }

  if(sysGlobalInt0 == needs)
  {
    sysGlobalInt3 = needs;
    sendMsgToInputs(itemOut);
  }

  if(itemOut >= 0.0)
  {
    Usererror("Failure");
    abort;
  }

  sending = FALSE;
}

// If the dialog data is inconsistent for simulation, abort.
on checkdata
{
  exec = sysGlobalint1; //block number for the executive block
  myIndex=sysGlobalint0; //assigns a unique integer for positioning in TimeArray and
  TimeBlockArray
  sysGlobalInt0 += 1; //updates for the next block

  //Ensure that the global array gaDT exists
  GaDTIndex = GaGetIndex("gaDT");
  If (GaDTIndex<0)
  {
    Usererror("The global array gaDT does not exist. This simulation must be run from VB");
    abort;
  }
}
}

```

```

on StepSize
{
  attribListIndex = GAGetIndex("_AttributeList");
  //check if toolID exists, if not add it to the attribute list
  numofAttribs = GAGetRowsByIndex (attribListIndex) ;
  if(GAFindStringAny(attribListIndex,"toolID",0,numofAttribs,6,FALSE) <0)
  {
    GAResizeByIndex(AttribListIndex,numofAttribs+1);
    GaSetString15("toolID",attribListIndex, numofAttribs, 0);
  }

  //check if toolType exists, if not add it to the attribute list
  numofAttribs = GAGetRowsByIndex (attribListIndex) ;
  if(GAFindStringAny(attribListIndex,"toolType",0,numofAttribs,8,FALSE) <0)
  {
    GAResizeByIndex(AttribListIndex,numofAttribs+1);
    GaSetString15("toolType",attribListIndex, numofAttribs, 0);
  }

  //check if itemType exists, if not add it to the attribute list
  numofAttribs = GAGetRowsByIndex (attribListIndex) ;
  if(GAFindStringAny(attribListIndex,"itemType",0,numofAttribs,8,FALSE) <0)
  {
    GAResizeByIndex(AttribListIndex,numofAttribs+1);
    GaSetString15("itemType",attribListIndex, numofAttribs, 0);
  }

  //check if MTBF exists, if not add it to the attribute list
  numofAttribs = GAGetRowsByIndex (attribListIndex) ;
  if(GAFindStringAny(attribListIndex,"MTBF",0,numofAttribs,4,FALSE) <0)
  {
    GAResizeByIndex(AttribListIndex,numofAttribs+1);
    GaSetString15("MTBF",attribListIndex, numofAttribs, 0);
  }

  //check if MTTR exists, if not add it to the attribute list
  numofAttribs = GAGetRowsByIndex (attribListIndex) ;
  if(GAFindStringAny(attribListIndex,"MTTR",0,numofAttribs,4,FALSE) <0)
  {
    GAResizeByIndex(AttribListIndex,numofAttribs+1);
    GaSetString15("MTTR",attribListIndex, numofAttribs, 0);
  }

  numofAttribs = GAGetRowsByIndex (attribListIndex) ;
}

// Initialize any simulation variables.
on initsim
{
  itemIndex = 0;
  rCount = -1; //row count for item index
  myNumber = MyBlockNumber();

  // get the pointer to the TimeArray and TimeEventMsgType arrays
  if(getPassedArray(SysGlobal0, timeArray) > 0)
  {
    // set the first event time to the start of the simulation
    timeArray[MyIndex] = StartTime;
    // get the pointer to the TimeBlocks array
    getPassedArray(SysGlobal7,TimeBlocks);
    // put this block's # in reserved position in TimeBlocks
    TimeBlocks[myindex] = myBlockNumber();
    //Get the pointer to the TimeEventMsgType array
    getPassedArray(SysGlobal13,TimeEventMsgType);
    //reserved position in TimeEventMsgType
    TimeEventMsgType[myIndex] = BlockReceive1Msg;
  }

  attribValueIndex = GaGetIndex("_AttribValues");
  GetSimulateMsgs(FALSE); //this is discrete event block. No simulate messages

  //find the output connector block number and store as downstreamNumber
  if (novalue(GetConnectedTextBlock(myNumber, getConNumber(myNumber,"ItemOut"))))
  {
    userError("Output not connected in block number " + myNumber + ".Recorrect");
    abort;
  }

  //Ensure downstream block is a queue block.
  if (GetConnectedTextBlock(myNumber, getConNumber(myNumber,"ItemOut"))<0)
  {
    GetConBlocks(myNumber, getConNumber(myNumber,"ItemOut"), intArray);
    If (getdimension(intArray)<=0)
  }
}

```

```

    {
    UserError("Output not connected in block number " + myNumber + ".Reconnect");
    abort;
    }

    downstreamBlockNumber = intArray[0][0];
    itemOutIsConnected = TRUE;

    if (GetBlockType(downstreamBlockNumber) != "Queues")
    {
    UserError("Item is blocked from leaving block number " + MyBlockNumber() + ". Place a
    queue after it prevent items from being destroyed.");
    itemOutIsConnected = FALSE;
    AbortAllSims();
    }
    }

    MakeArray(DTInfo,gaGetRowsbyIndex(gaDTIndex)); //Expand the array

    For (i=0;i<getDimension(DTInfo);i++)
    {
    For (j=0;j<getDimensioncolumns(DTInfo);j++)
    DTInfo[i][j]=gaGetReal(gaDTIndex,i,j);
    }

    DynamicDataTable(myNumber, "DTTable", DTInfo);
    //Find the col location of ToolID, ToolType, ItemType, dtType, dtNum.
    numofAttribs = GAGetRowsByIndex (attribListIndex) ;
    if(GAFindStringAny(attribListIndex,"toolID",0,numofAttribs,6,FALSE) >=0)
    {
    toolIDCol = 1+ GAFindStringAny(attribListIndex,"toolID",0,numofAttribs,6,FALSE);
    }

    if(GAFindStringAny(attribListIndex,"toolType",0,numofAttribs,8,FALSE) >=0)
    {
    toolTypeCol = 1+ GAFindStringAny(attribListIndex,"toolType",0,numofAttribs,8,FALSE);
    }

    if(GAFindStringAny(attribListIndex,"itemType",0,numofAttribs,8,FALSE) >=0)
    {
    itemTypeCol = 1+ GAFindStringAny(attribListIndex,"itemType",0,numofAttribs,8,FALSE);
    }

    if(GAFindStringAny(attribListIndex,"MTBF",0,numofAttribs,4,FALSE) >=0)
    {
    MTBFCol = 1+ GAFindStringAny(attribListIndex,"MTBF",0,numofAttribs,4,FALSE);
    }

    if(GAFindStringAny(attribListIndex,"MTTR",0,numofAttribs,4,FALSE) >=0)
    {
    MTTRCol = 1+ GAFindStringAny(attribListIndex,"MTTR",0,numofAttribs,4,FALSE);
    }

    TimeArray[myIndex] = startTime; //so that we get a block recieve1 message at time zero
}

//we receive this message from the executive because we posted and event
on BlockReceive1
{
    If (currentTime == StartTime)
    {
    For (i=0; i<getdimension(dtInfo);i++) //loop to ensure we get a BlockRecieve message until
    all tools are released
    {
    ItemOut = CreateItem(); // create an item and set the output to its index value
    SendItem(); // initiate the send item procedure
    }
    }
}
}

```

B.4.4 Preventative maintenance generator code

The preventative maintenance generator block reads from the *gaPM* array the mean time before failure and the mean time to repair parameters of each tool. It then creates the

downtime *items*, assigns them the MTBF and MTTR values as attributes and releases them into the model at simulation time zero.

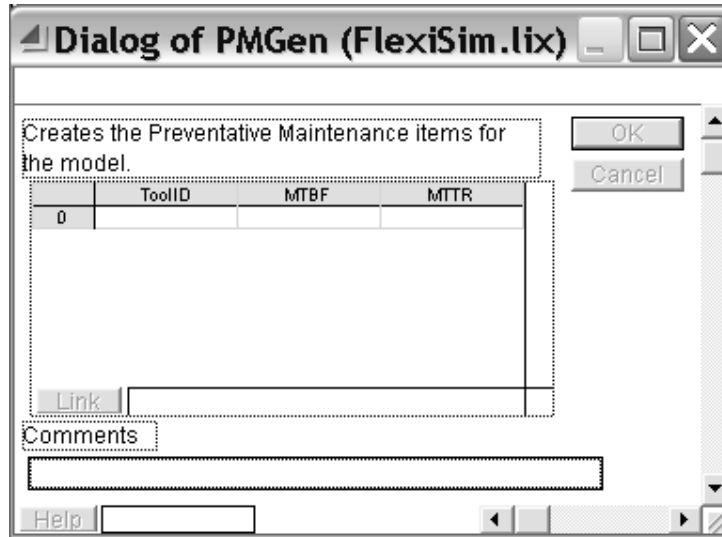


Figure B.7: Dialog of custom preventative maintenance generator block for FTM Application.

```

// Constants
real   itemArrayR[][3];
integer itemArrayI[][5];
integer itemArray3D[][10];
string itemArrayT[];
real   itemArrayC[][10];
real   timeArray[];
integer timeblocks[];
integer timeEventMsgType[]; // array that stores the message that this block will receive
                               when a message occurs
integer exec;
integer myIndex;
integer myNumber;
integer rCount; //number of rows in the itemArrays
integer intArray[][2]; //the connected outputs
integer downstreamBlockNumber;
integer itemOutisConnected;
integer i,j;
integer ItemIndex;
integer itemTypeCol;
integer toolTypeCol;
integer MTBFCol;
integer MTTRCol;
integer toolIDCol;
integer attribValueIndex;
integer attribListIndex;
integer numOfAttribs;
integer sending;
integer gaPMIndex;
integer pmInfoIndex;
real PMInfo[][3];

// item messages
constant rejects is 0;
constant wants is 1;
constant taken is 2;
constant needs is 3;
constant query is 4;
constant notify is 5;
constant blocked is 6;
constant init is 7;

```

```

//this procedure takes in all the itemArray information controlled by the executive
procedure getArrays()
{
  if (rCount != sysGlobalint2)
  {
    getPassedArray(sysGlobal3, itemArrayR);
    getPassedArray(sysGlobal4, itemArrayI);
    getPassedArray(sysGlobal9, itemArrayC);
    getPassedArray(sysGlobal12, itemArray3D);
    rCount = sysGlobalint2;
  }
}

//this function is called to create an item by interacting with executive
integer createItem()
{
  getArrays();
  if (sysGlobalint1+1 >= sysGlobalint2) //the first free row of the item index list
  {
    sendMsgToBlock(exec, blockreceive0Msg); // expand array if not big enough
    getArrays();
  }

  ItemIndex = sysGlobalint1; //the value is now the items index number
  itemArrayI[ItemIndex][0] = 0; // row used
  itemArrayI[ItemIndex][1] = 0; // batch ID
  itemArrayI[ItemIndex][2] = 0; // user value
  itemArrayI[ItemIndex][3] = 0; // batch ID #2
  itemArrayI[ItemIndex][4] = 0; // unused (not sure about this says, manual says its block
    number where item is
  itemArrayR[ItemIndex][0] = 1.0; // value
  itemArrayR[ItemIndex][1] = BLANK; // Priority
  itemArrayR[ItemIndex][2] = BLANK; // user value

  //update the attribute values.
  GaSetReal(pmInfo[i][0], attribValueIndex, itemIndex, toolIDCol);
  GaSetReal(pmInfo[i][1], attribValueIndex, itemIndex, MTBFCol);
  GaSetReal(pmInfo[i][2], attribValueIndex, itemIndex, MTTRCol);
  GaSetReal(4, attribValueIndex, ItemIndex, itemTypeCol); //itemType is 4 because it is a PM item

  GaSetReal(66, attribValueIndex, ItemIndex, 0); //0 for animation column, 65 for green circle

  sendMsgToBlock(exec, blockReceive1Msg); // check for next space
  return(ItemIndex);
}

//This function was called from 'on blockReceive1'
procedure SendItem()
{
  sending = TRUE;
  if(itemOut > 0.0) //if an item is ready to leave
  {
    sysGlobalInt3 = wants; //tell upstreamblock we are ready
    sendMsgToInputs(itemOut); //upstream will check if it can accept
  }

  if(sysGlobalInt0 == needs)
  {
    sysGlobalInt3 = needs;
    sendMsgToInputs(itemOut);
  }

  if(itemOut >= 0.0)
  {
    Usererror("Failure");
    abort;
  }

  sending = FALSE;
}

// If the dialog data is inconsistent for simulation, abort.
on checkdata
{
  exec = sysGlobalint1; //block number for the executive block
  myIndex=sysGlobalint0; //assigns a unique integer for positioning in TimeArray and
    TimeBlockArray
  sysGlobalInt0 += 1; //updates for the next block

  //Ensure that the global array gaDT exists
  GaPMIndex = GaGetIndex("gaPM");
  If (GaPMIndex<0)
  {

```

```

    Usererror("The global array gaPM does not exist. This simulation must be run from VB");
    abort;
}
}
on StepSize
{
    attribListIndex = GAGetIndex("_AttributeList");
    //check if toolID exists, if not add it to the attribute list
    numofAttribs = GAGetRowsByIndex (attribListIndex) ;
    if(GAFindStringAny(attribListIndex, "toolID", 0, numofAttribs, 6, FALSE) <0)
    {
        GAResizeByIndex (AttribListIndex, numofAttribs+1);
        GaSetString15("toolID", attribListIndex, numofAttribs, 0);
    }

    //check if toolType exists, if not add it to the attribute list
    numofAttribs = GAGetRowsByIndex (attribListIndex) ;
    if(GAFindStringAny(attribListIndex, "toolType", 0, numofAttribs, 8, FALSE) <0)
    {
        GAResizeByIndex (AttribListIndex, numofAttribs+1);
        GaSetString15("toolType", attribListIndex, numofAttribs, 0);
    }

    //check if itemType exists, if not add it to the attribute list
    numofAttribs = GAGetRowsByIndex (attribListIndex) ;
    if(GAFindStringAny(attribListIndex, "itemType", 0, numofAttribs, 8, FALSE) <0)
    {
        GAResizeByIndex (AttribListIndex, numofAttribs+1);
        GaSetString15("itemType", attribListIndex, numofAttribs, 0);
    }

    //check if MTBF exists, if not add it to the attribute list
    numofAttribs = GAGetRowsByIndex (attribListIndex) ;
    if(GAFindStringAny(attribListIndex, "MTBF", 0, numofAttribs, 4, FALSE) <0)
    {
        GAResizeByIndex (AttribListIndex, numofAttribs+1);
        GaSetString15("MTBF", attribListIndex, numofAttribs, 0);
    }

    //check if MTTR exists, if not add it to the attribute list
    numofAttribs = GAGetRowsByIndex (attribListIndex) ;
    if(GAFindStringAny(attribListIndex, "MTTR", 0, numofAttribs, 4, FALSE) <0)
    {
        GAResizeByIndex (AttribListIndex, numofAttribs+1);
        GaSetString15("MTTR", attribListIndex, numofAttribs, 0);
    }

    numofAttribs = GAGetRowsByIndex (attribListIndex) ;
}

// Initialize any simulation variables.
on initsim
{
    itemIndex = 0;
    rCount = -1; //row count for item index
    myNumber = MyBlockNumber();

    // get the pointer to the TimeArray and TimeEventMsgType arrays
    if(getPassedArray(SysGlobal0, timeArray) > 0)
    {
        // set the first event time to the start of the simulation
        timeArray[MyIndex] = StartTime;
        // get the pointer to the TimeBlocks array
        getPassedArray(SysGlobal7, TimeBlocks);
        // put this block's # in reserved position in TimeBlocks
        TimeBlocks[myindex] = myBlockNumber();
        //Get the pointer to the TimeEventMsgType array
        getPassedArray(SysGlobal13, TimeEventMsgType);
        //reserved position in TimeEventMsgType
        TimeEventMsgType[myIndex] = BlockReceiveMsg;
    }

    attribValueIndex = GaGetIndex("_AttribValues");
    GetSimulateMsgs(FALSE); //this is discrete event block. No simulate messages

    //find the output connector block number and store as downstreamNumber
    if (novalue(GetConnectedTextBlock(myNumber, getConNumber(myNumber, "ItemOut"))))
    {
        userError("Output not connected in block number " + myNumber + ".Reconnect");
        abort;
    }
}

```

Appendix B. FTM Application Code

```

//Ensure downstream block is a queue block.
if (GetConnectedTextBlock(myNumber, getConNumber(myNumber, "ItemOut"))<0)
{
  GetConBlocks(myNumber, getConNumber(myNumber, "ItemOut"), intArray);

  If (getdimension(intArray)<=0)
  {
    UserError("Output not connected in block number " + myNumber + ".Reconnect");
    abort;
  }

  downstreamBlockNumber = intArray[0][0];
  itemOutIsConnected = TRUE;

  if (GetBlockType(downstreamBlockNumber) != "Queues")
  {
    UserError("Item is blocked from leaving block number " + MyBlockNumber() + ". Place a
      queue after it prevent items from being destroyed.");
    itemOutIsConnected = FALSE;
    AbortAllSims();
  }
}

MakeArray(PMInfo, gaGetRowsbyIndex(gaPMIndex)); //Expand the array

For (i=0; i<getDimension(PMInfo); i++)
{
  For (j=0; j<getDimensioncolumns(PMInfo); j++)
  PMInfo[i][j]=gaGetReal(gaPMIndex, i, j);
}

DynamicDataTable(myNumber, "PMDataTable", PMInfo);

//Find the col location of ToolID, ToolType, ItemType, dtType, dtNum.
numofAttribs = GAGetRowsByIndex (attribListIndex) ;

if (GAFindStringAny(attribListIndex, "toolID", 0, numofAttribs, 6, FALSE) >=0)
{
  toolIDCol = 1+ GAFindStringAny(attribListIndex, "toolID", 0, numofAttribs, 6, FALSE);
}

if (GAFindStringAny(attribListIndex, "toolType", 0, numofAttribs, 8, FALSE) >=0)
{
  toolTypeCol = 1+ GAFindStringAny(attribListIndex, "toolType", 0, numofAttribs, 8, FALSE);
}

if (GAFindStringAny(attribListIndex, "itemType", 0, numofAttribs, 8, FALSE) >=0)
{
  itemTypeCol = 1+ GAFindStringAny(attribListIndex, "itemType", 0, numofAttribs, 8, FALSE);
}

if (GAFindStringAny(attribListIndex, "MTBF", 0, numofAttribs, 4, FALSE) >=0)
{
  MTBFCol = 1+ GAFindStringAny(attribListIndex, "MTBF", 0, numofAttribs, 4, FALSE);
}

if (GAFindStringAny(attribListIndex, "MTTR", 0, numofAttribs, 4, FALSE) >=0)
{
  MTTRCol = 1+ GAFindStringAny(attribListIndex, "MTTR", 0, numofAttribs, 4, FALSE);
}

TimeArray[myIndex] = startTime; //so that we get a block recieve1 message at time zero
}

//we receive this message from the executive because we posted and event
on BlockReceive1
{
  If (currentTime == StartTime)
  {
    For (i=0; i<getdimension(pmInfo); i++) //loop to ensure we get a BlockRecieve message
      until all tools are released
    {
      ItemOut = CreateItem(); // create an item and set the output to its index value
      SendItem(); // initiate the send item procedure
    }
  }
}

```

B.4.5 Pairing block code

The pairing block is responsible for storing any of the necessary tool, lot, PM or downtime *items* to be paired and released signifying the occurrence of some event.

```

//Static Variables:
integer myIndex;
integer exec;
real timeArray[];
integer timeBlocks[];
integer itemArrayI[][5];
integer itemArray3D[][10];
real nextTime;
real lastTime;
real beginTime;
integer delayTimeInConnected;
integer itemIndex;
integer sending;
integer getting;
integer meBlocked;
integer gotItem;
real timeUnitConversionFactor;
integer rescheduled;
integer connectedToSensor;
integer itemOutConnectionInfo[][2];
integer numInBlocks;
integer inBlock;
integer inConn;
integer connected[][2];
real clearStatsTime;
real lastTConnectorVal;
real utilSum;
integer colClicked;
integer rowClicked;
integer costAttrib;
integer rateAttrib;
integer numOfAttribs;
integer opIDAttribCol;
integer toolIDAttribCol;
integer toolTypeAttribCol;
integer itemTypeAttribCol;
integer lotList;
integer toolList;
integer offlineList;
real LotTableInfo[][3];
real ToolTableInfo[][4];
real OfflineTableInfo[][4];

//attribute variables
integer attribListIndex; //index of the array holding the attrib list
integer attribValuesIndex; //index to global array containing the attribute values
integer attribTypeIndex; //index of the _attribType GA
string attribCategories[][3]; //used to separate the list of all attrib names into three
//columns: normal attribs, string attribs, db attribs
string31 attribNameArraysRemote[];
string15 attribNameOldNew[];

//btb: common variables
integer BTB_AnimationIndex;
integer BTB_NumUserPicts;
integer BTB_NumPicts;
string31 BTB_PictNames[];

//btb: "change to" animation vars
string31 BTB_ChangeToPicture;
//btb: multi-out options table vars
integer BTB_CurrentNumItemOutCons;
string31 BTB_MultiOutOptionsTable_DA[][2];
string BTB_MultiOutOptions[];
integer BTB_MultiOutNumEquivs[][2];
//col 0 contains the numeric equivalent of which option has been chosen for that row
//col 1 contains the numeric equivalent of which picture has been chosen for that row
//btb: attribute based conversion
integer BTB_AttribType[];
string BTB_AttribPopContents[];
string15 BTB_AttribNamesChosen[];

```



```

integer BTB_AttribColumnID[];
string31 BTB_AttribTable_DA[][5];
integer BTB_AttribNumConvert;
real BTB_AttribConvert[][5];
    //col 0 contains the different attribute values that items may have
    //col 1 contains the numeric equivalents of the pictures chosen in the picture table
integer BTB_NumRowsAttribTable;

// 3D animation
String31 E3DSelectedNames[];
String31 E3DNames[];
integer E3DObject[];
integer UnmountDirection;
integer E3D_AnimationIndex;
String31 E3DObjects[];
integer SaveXYClickPosition[];
real B3DPictureAngles[];
String31 Skins[];
String31 E3DChangeObjectName[];
integer E3DChangeObjectNum[];
integer E3D_AnimationItemIndex;    // index of global array with item based animation objects

// Proof Animation
integer ProofStuff[];
String31 ProofStr[];    // contains animation object and attribute name

//Constants:
constant OUTPUT is 0;
constant INPUT is 1;
constant REJECTS is 0;
constant WANTS is 1;
constant TAKEN is 2;
constant NEEDS is 3;
constant QUERY is 4;
constant NOTIFY is 5;
constant CONDITIONAL_ROUTE_BLOCKER is 6;
constant PROPERTY_NAME_CHECK is 7;
constant ITEM_IN is 0;
constant ITEM_OUT is 1;
constant PUSH is 0;
constant PULL is 1;
constant NO_RESPONSE is -1;

//animation constants
constant BTB_ANIM_OBJECT is 1;

//proof animation constants
constant ProofControlRow is 0;    //block number of the proof control block (this block)
constant ConcurrentRow is 1;    // true if concurrent animation is turned on
constant TraceRow is 2;    // true if the trace option is turned on
constant FileNumRow is 3;    // file number for the trace file
constant PauseRow is 4;    // true if proof has paused the simulation
constant PausingBlock is 5;    // block number of the pausing block
constant BlockNumber is 6;    // block number of the block which is sending a message to
    the control block
constant ItemIndexRow is 7;    // item index of the animated item
constant CommandType is 8;    // animation command associated with this block

//prototypes
procedure GetItem();
procedure SendItem(integer sendMechanism);
procedure ProofAnimate(integer ProofIndex, Real SendValue);
procedure UpdateList(integer theIndex);
procedure CheckOffline();
procedure SendItem(integer theIndex);
procedure CheckLots();
procedure UpdateLotTable();
procedure UpdateToolTable();
procedure UpdateOfflineTable();
procedure changeLastProcessed(integer theArrayIndex, integer thePos);
procedure shuffleTiedOps(integer tempToolInfoIndex);

//Include files:
#include "Constants v7.h"
#include "ColumnTags v7.h"
#include "Attribs v7.h"
#include "BTB Animation v7.h"
#include "Proof v7.h"

//this procedure re-establishes the location in memory of itemArrayI and keeps track of how
    many items are in the model.
on DEExecutiveArrayResize
{

```

```

GetPassedArray(sysGlobal4, itemArrayI);
GetPassedArray(sysGlobal12, ItemArray3D);
}

// This routine sends out a message, and returns an answer.
integer SendMsg(integer whatMsg, integer where)
{
  sysGlobalint3 = whatMsg;

  if (where == INPUT)
    SendMsgToOutputs(itemIn);
  else
    SendMsgToInputs(itemOut);
  return(sysGlobalint0);
}

procedure UpdateLotTable()
{
  integer i;

  MakeArray(LotTableInfo, ListGetElements(MyBlockNumber(), LotList));
  for (i=0;i<ListGetElements(MyBlockNumber(), LotList);i++)
  {
    LotTableInfo[i][0]= ListGetDouble(MyBlockNumber(), LotList, i, 0);
    LotTableInfo[i][1]= ListGetLong(MyBlockNumber(), LotList, i, 0);
    LotTableInfo[i][2]= ListGetLong(MyBlockNumber(), LotList, i, 1);
  }

  DynamicDataTable(MyBlockNumber(), "LotTable", LotTableInfo);
  RefreshDatatableCells(MyBlockNumber(), "LotTable", 0, 0, ListGetElements(MyBlockNumber(),
    LotList)-1, 2);
}

procedure UpdateToolTable()
{
  integer i;

  MakeArray(ToolTableInfo, ListGetElements(MyBlockNumber(), ToolList));
  for (i=0;i<ListGetElements(MyBlockNumber(), ToolList);i++)
  {
    ToolTableInfo[i][0]= ListGetDouble(MyBlockNumber(), ToolList, i, 0);
    ToolTableInfo[i][1]= ListGetLong(MyBlockNumber(), ToolList, i, 0);
    ToolTableInfo[i][2]= ListGetLong(MyBlockNumber(), ToolList, i, 1);
    ToolTableInfo[i][3]= ListGetLong(MyBlockNumber(), ToolList, i, 2);
  }

  DynamicDataTable(MyBlockNumber(), "ToolTable", ToolTableInfo);
  RefreshDatatableCells(MyBlockNumber(), "ToolTable", 0, 0, ListGetElements(MyBlockNumber(),
    ToolList)-1, 3);
}

procedure UpdateOfflineTable()
{
  integer i;
  MakeArray(OfflineTableInfo, ListGetElements(MyBlockNumber(), OfflineList));
  for (i=0;i<ListGetElements(MyBlockNumber(), OfflineList);i++)
  {
    OfflineTableInfo[i][0]= ListGetDouble(MyBlockNumber(), OfflineList, i, 0);
    OfflineTableInfo[i][1]= ListGetLong(MyBlockNumber(), OfflineList, i, 0);
    OfflineTableInfo[i][2]= ListGetLong(MyBlockNumber(), OfflineList, i, 1);
    OfflineTableInfo[i][3]= ListGetLong(MyBlockNumber(), OfflineList, i, 2);
  }

  DynamicDataTable(MyBlockNumber(), "OfflineTable", OfflineTableInfo);
  RefreshDatatableCells(MyBlockNumber(), "OfflineTable", 0, 0, ListGetElements(MyBlockNumber()
    , OfflineList)-1, 3);
}

//GetItem() is the procedure responsible for getting items from the upstream residence block.
procedure GetItem()
{
  integer i;
  getting = TRUE; //set getting flag to avoid re-entrance.
  if (itemIn > 0.0) //if item is available on item in, pull it in
  {
    itemIndex = itemIn;
    itemIn = -itemIn;
    BTB_GetItem(itemIndex, BTB_ANIM_OBJECT, inBlock, inConn, 0); //0 for index of itemIn
    connector
    ItemArrayI[itemIndex][4] = MyBlockNumber(); //record where this item is in the integer
    item array
    SendMsg(TAKEN, INPUT); //item taken
    ConnectorMsgBreak(); //if this activity was placed in parallel, then prevent the other

```

```

        parallel activities from getting messages from upstream
    }
    else
    {
        userError("An upstream block sent or returned 'needs' without making an item available on
            " +
            "its itemOut connector. Currently in 'GetItem()' of block " + MyBlockNumber() + ".")
        );
        abort;
    }

    UpdateList(itemIndex);
    CheckOffline();
    CheckLots();
    getting = FALSE;
}

procedure UpdateList(integer theIndex)
{
    integer i;
    integer tempItemType;
    integer tempToolDetailIndex;

    //get the itemType
    tempItemType = gaGetReal(attribValuesIndex,theIndex,itemTypeAttribCol);

    switch (tempItemType)
    {
        case 1:
            //item is a lot.
            ListCreateElement(myblockNumber(), LotList);
            ListSetLong(myBlockNumber(), LotList, -1, 0, gaGetReal(attribValuesIndex,theIndex,
                opIDAttribCol)); // set opID
            ListSetLong(myBlockNumber(), LotList, -1, 1, gaGetReal(attribValuesIndex,theIndex,
                itemTypeAttribCol)); // set itemType
            ListSetLong(myBlockNumber(), LotList, -1, 2, theIndex); // set itemIndex
            ListSetDouble(myBlockNumber(), LotList, -1, 0, currentTime); // set entryTime
            ListAddElement(myblockNumber(), LotList, -2);
            UpdateLotTable();
            break;

        case 2:
            //item is a tool
            tempToolDetailIndex = GAGetIndex("gaOpRank"+gaGetReal(attribValuesIndex,theIndex,
                toolIDAttribCol));
            ListCreateElement(myblockNumber(), ToolList);
            ListSetLong(myBlockNumber(), ToolList, -1, 0, gaGetReal(attribValuesIndex,theIndex,
                toolIDAttribCol)); // set toolID
            ListSetLong(myBlockNumber(), ToolList, -1, 1, gaGetReal(attribValuesIndex,theIndex,
                toolTypeAttribCol)); // set toolType
            ListSetLong(myBlockNumber(), ToolList, -1, 2, gaGetReal(attribValuesIndex,theIndex,
                itemTypeAttribCol)); // set itemType
            ListSetLong(myBlockNumber(), ToolList, -1, 3, tempToolDetailIndex); // find index to tool
                details
            ListSetLong(myBlockNumber(), ToolList, -1, 4, theIndex); // set itemIndex
            ListSetDouble(myBlockNumber(), ToolList, -1, 0, currentTime); // set entryTime
            ListAddElement(myblockNumber(), ToolList, -2);
            UpdateToolTable();
            break;

        case 3:
            //item is a downtime (uncheduled) event
            ListCreateElement(myblockNumber(), offlineList);
            ListSetLong(myBlockNumber(), offlineList, -1, 0, gaGetReal(attribValuesIndex,theIndex,
                toolIDAttribCol)); // set toolID
            ListSetLong(myBlockNumber(), offlineList, -1, 1, gaGetReal(attribValuesIndex,theIndex,
                toolTypeAttribCol)); // set toolType
            ListSetLong(myBlockNumber(), offlineList, -1, 2, gaGetReal(attribValuesIndex,theIndex,
                itemTypeAttribCol)); // set itemType
            ListSetLong(myBlockNumber(), offlineList, -1, 3, theIndex); // set itemIndex
            ListSetDouble(myBlockNumber(), offlineList, -1, 0, currentTime); // set entryTime
            ListAddElement(myblockNumber(), offlineList, -2);
            UpdateOfflineTable();
            break;

        case 4:
            //item is a PM item
            ListCreateElement(myblockNumber(), offlineList);
            ListSetLong(myBlockNumber(), offlineList, -1, 0, gaGetReal(attribValuesIndex,theIndex,
                toolIDAttribCol)); // set toolID
            ListSetLong(myBlockNumber(), offlineList, -1, 1, gaGetReal(attribValuesIndex,theIndex,
                toolTypeAttribCol)); // set toolType
            ListSetLong(myBlockNumber(), offlineList, -1, 2, gaGetReal(attribValuesIndex,theIndex,

```

```

        itemTypeAttribCol)); // set itemType
ListSetLong(myBlockNumber(), offlineList, -1, 3, theIndex); // set itemIndex
ListSetDouble(myBlockNumber(), offlineList, -1, 0, currentTime); // set entryTime
ListAddElement(myBlockNumber(), offlineList, -2);
UpdateOfflineTable();
break;

default: // any other number
UserError("Spurious ItemType found in "+myblocknumber()+".");
abort;
break;
}
}

procedure CheckOffline()
{
    integer i, j;

restart:
for (i=0; i<ListGetElements(myBlockNumber(), ToolList); i++)
{
    for (j=0; j<ListGetElements(myBlockNumber(), offlineList); j++)
    {
        If (ListGetLong(myBlockNumber(), ToolList, i, 0) == ListGetLong(myBlockNumber(),
            offlineList, j, 0))
        {
            //Remove the Tool from the ToolList and send it using SendItem
            SendItem(ListGetLong(myBlockNumber(), ToolList, i, 4));
            ListDeleteElement(myBlockNumber(), ToolList, i);
            UpdateToolTable();

            //Remove the Offline Event from the list and send it using Send Item
            SendItem(ListGetLong(myBlockNumber(), offlineList, j, 3));
            ListDeleteElement(myBlockNumber(), offlineList, j);
            UpdateOfflineTable();
            goto Restart;
        }
    }
}
}

procedure CheckLots()
{
    integer i, j, k, m;
    integer tempToolInfoIndex;
    integer theOperation;
    integer tempVal;

restart:
for (i=0; i<ListGetElements(myBlockNumber(), ToolList); i++)
{
    tempToolInfoIndex = ListGetLong(myBlockNumber(), ToolList, i, 3);

    switch (GAGetReal(tempToolInfoIndex, 0, 1))
    //row0, col1 contains an indication of the priority system of the tool.
    // -1: last processed
    // 0: no Ranking (FIFO)
    // any other number: user selected ranking system
    {
        case -1:
        {
            //use last processed. Tied Operation are selected on a FIFO basis
            for (j=0; j<GAGetRowsByIndex(tempToolInfoIndex); j++)
            {
                theOperation = GAGetReal(tempToolInfoIndex, j, 2); //converts it to integer, last col
                    stores last selected

                for (k=0; k<ListGetElements(myBlockNumber(), LotList); k++)
                {
                    If (ListGetLong(myBlockNumber(), LotList, k, 0) == theOperation)
                    {
                        //Remove the Tool from the ToolList and send it using SendItem
                        SendItem(ListGetLong(myBlockNumber(), ToolList, i, 4));
                        ListDeleteElement(myBlockNumber(), ToolList, i);
                        UpdateTooltable();

                        //Remove the Lot from the list and send it using Send Item
                        SendItem(ListGetLong(myBlockNumber(), LotList, k, 2));
                        ListDeleteElement(myBlockNumber(), LotList, k);
                        UpdateLotTable();
                        changeLastProcessed(tempToolInfoIndex, j);
                        goto restart;
                    }
                }
            }
        }
    }
}
}

```



```

    if (GAGetReal(theArrayIndex, i, 1) == GAGetReal(theArrayIndex, i+1, 1))
    {
        if (Random(100)>=49) // 50/50 chance of swapping them
        {
            //swap them
            temp = GAGetReal(theArrayIndex, i, 2);
            GASetReal(GAGetReal(theArrayIndex, i+1, 2), theArrayIndex, i, 2);
            GASetReal(temp, theArrayIndex, i+1, 2);
        }
    }
}

procedure SendItem(integer theIndex)
{
    sending = TRUE;
    itemIndex = theIndex;

    //if appropriate, move current item to itemOut, i.e., if have an
    if(itemIndex > 0)
    {
        if(itemOut <= 0.0)
        {
            itemOut = itemIndex; //move the item out of processing
            itemIndex = 0; //signal we're no longer processing an item
        }
        else
        {
            UserError("An item message passing failure occurred in SendItem() of " + "block number "
                + MyBlockNumber() + ".");
            abort;
        }
    }

    if(itemOut > 0.0) //if an item is ready to leave
    {
        if(sendMsg(WANTS, OUTPUT) == NEEDS) //if downstream block wants the item, it will return
            NEEDS
        {
            BTB_SendItem(itemOut, 0); //set the btb animation attrib on the item
            sendMsg(NEEDS, OUTPUT);

            if(itemOut < 0.0) //if the downstream block took the item
            {
                meBlocked = FALSE;
            }
            else
            {
                UserError("An item message passing failure occurred in SendItem() of " +
                    "block number " + MyBlockNumber() + ". The downstream block returned " +
                    "NEEDS but then did not take the item.Use a queue block here");
                abort;
            }
        }
        else //if downstream block does not want the item, then the item is blocked
        {
            meBlocked = TRUE;
        }
    }
    else
    {
        UserError("An item message passing failure occurred in SendItem() of " + "block number " +
            MyBlockNumber() + ". An item that was done processing has disappeared.");
        abort;
    }

    sending = FALSE;
}

//called in itemOut after receiving a taken message
procedure Departure()
{
    if(connectedToSensor == TRUE) //if itemOut is connected to a block containing a sensor
        connector (eg, the timer, gate, and status blocks), then send a notify message
        sendMsg(NOTIFY, OUTPUT);
    meBlocked = FALSE;
}

on itemIn
{
    integer whichMessage;

    whichMessage = sysGlobalint3;
}

```

```

if (whichMessage == CONDITIONAL_ROUTE_BLOCKER || whichMessage == NOTIFY)
    return;
else if ( (whichMessage == WANTS || whichMessage == NEEDS) && (getting) )    //safety
    checking
    {
        sysGlobalInt0 = REJECTS;
    }
else if (whichMessage == WANTS)

    {
        sysGlobalInt0 = NEEDS;
        ConnectorMsgBreak();    //cancels the messages to other blocks connected to the output of
            this block
    }

else if (whichMessage == NEEDS)
    {
        GetItem();    //upstream block is pushing an item to us.
        sysGlobalInt0 = REJECTS;
        connectorMsgBreak();
    }
else
    {
        userError("Spurious message received in 'on itemIn' in block " + MyBlockNumber() + ".");
        abort;
    }

sysGlobalInt3 = whichMessage;
    //sending message out to other blocks could have resulted in sysGlobalInt3 being
    //changed so before returning, set sysGlobalInt3 to its original state.
}

on itemOut
{
    integer whichMessage;
    whichMessage = sysGlobalInt3;

    if (whichMessage == QUERY)    //a downstream block wants to know what the item index of the
        next item will be.
        {
            if (itemOut > 0.0)
                {
                    sysGlobalInt0 = itemOut;
                }
            else
                {
                    sysGlobalInt0 = 0;
                }
        }

    else if (whichMessage == WANTS)    //a downstream block is trying to pull an item.
        {
        }

    else if (whichMessage == TAKEN)
        {
        }

    else
        {
            userError("Spurious message received in 'on ItemOut' in block " + MyBlockNumber() + ".");
            abort;
        }

    sysGlobalInt3 = whichMessage;
        //sending message out to other blocks could have resulted in sysGlobalInt3 being
        //changed so before returning, set sysGlobalInt3 to its original state.
}

//If the dialog data is inconsistent for simulation, abort.
on checkData
{
    exec = sysGlobalInt1;    // the id number of the exec block
    myIndex = sysGlobalInt0;
    sysGlobalInt0 += 1;
}

// Initialize any simulation variables.
on initSim
{
    integer x, y, i;
    integer numBlocksConnectedToItemOut;
}

```

```

string nextConnName;
getting = FALSE;
sending = FALSE;
itemOut = 0.0;
itemIndex = 0;
meBlocked = FALSE;
gotItem = FALSE;
lastTConnectorVal = 0.0;
rescheduled = FALSE;

attribValuesIndex = GaGetIndex("_AttribValues");

//inBlock and inConn are used in block to block animation.
inBlock = 0;
inConn = 0;
GetConBlocks(MyBlockNumber(), ITEM_IN, connected);
//The dynamic array "connected" is now filled with all the global block
//numbers and connector numbers of the "net list" that itemIn is part of.
if (GetDimension(connected) > 0) //itemIn is connected
{
    inBlock = connected[0][0]; //the global block number of the block directly connected to
    itemIn
    inConn = connected[0][1]; //the connector number of the connector directly connected
    to itemIn
}
DisposeArray(connected);

//find all the blocks connected to itemOut and check to see if any connectors are named "
sensor"
connectedToSensor = FALSE;
numBlocksConnectedToItemOut = GetConBlocks(MyBlockNumber(), ITEM_OUT, itemOutConnectionInfo)
; //for blocks connected to itemOut, put block and connector number info into the "
itemOutConnectionInfo" array
for(i=0; i<numBlocksConnectedToItemOut; i++)
{
    nextConnName = GetConName(itemOutConnectionInfo[i][0], itemOutConnectionInfo[i][1]);
    if(nextConnName == "sensorIn")
    {
        connectedToSensor = TRUE;
        break;
    }
}

DisposeArray(itemOutConnectionInfo);

if ( getPassedArray(sysGlobal0, timeArray) )
{
    //do nothing
}
else
{
    userError ("The Executive block must be present and to the left of all blocks on the
    worksheet");
    abort;
}

getPassedarray(sysGlobal7,timeblocks);
timeblocks[myindex] = myBlockNumber();
getSimulateMsgs(FALSE);

if(HAnim && !NoValue(HAnimNum) && HanimNum > 0.999)
{
    animationHide(-HAnimNum, FALSE);
}

//Find the column location of OpID, ToolID, ToolType and ItemType
numofAttribs = GAGetRowsByIndex (attribListIndex) ;

if(GAFindStringAny(attribListIndex, "opID",0,numofAttribs,4,FALSE) >=0)
{
    opIDAttribCol = 1+ GAFindStringAny(attribListIndex, "opID",0,numofAttribs,4,FALSE);
}

if(GAFindStringAny(attribListIndex, "toolID",0,numofAttribs,6,FALSE) >=0)
{
    toolIDAttribCol = 1+ GAFindStringAny(attribListIndex, "toolID",0,numofAttribs,6,FALSE);
}

if(GAFindStringAny(attribListIndex, "toolType",0,numofAttribs,8,FALSE) >=0)
{
    toolTypeAttribCol = 1+ GAFindStringAny(attribListIndex, "toolType",0,numofAttribs,8,FALSE);
}

```



```

if(GAFindStringAny(attribListIndex, "itemType", 0, numofAttribs, 8, FALSE) >=0)
{
    itemTypeAttribCol = 1+ GAFindStringAny(attribListIndex, "itemType", 0, numofAttribs, 8, FALSE);
}

//Create lotlist with the following cols opID, ItemType, itemIndex and EntryTime sorted by
//EntryTime
LotList = ListCreate(myblockNumber(), 3, 1, 0, 0, 0, 1, 0);
//Create toolList with the following cols toolID, toolType, itemType, IndexToToolDetails,
//itemIndex and EntryTime sorted by EntryTime
ToolList = ListCreate(myblockNumber(), 5, 1, 0, 0, 0, 1, 0);
//Create OfflineList with the following cols toolID, toolType, itemType, itemIndex and
//EntryTime sorted by EntryTime
OfflineList = ListCreate(myblockNumber(), 4, 1, 0, 0, 0, 1, 0);
}

on DialogClick
{
    string31 dName; // name of the dialog item clicked

    dName = whichDialogItemClicked();
    rowClicked = WhichDTCellClicked(0);
    colClicked = WhichDTCellClicked(1);
}

on DataTableResize
{
    integer numRowsAttribTable;
    string whichTable;

    whichTable = WhichDialogItemClicked();
    if(whichTable == "BTB_Attrib_ttbl")
    {
        numRowsAttribTable = GetDimension(BTB_AttribTable_DA);
        numRowsAttribTable = NumericParameter("Maximum rows in attribute animation conversion
        table:", numRowsAttribTable);
        BTB_OnDataTableResize(numRowsAttribTable);
    }
}

on FinalCalc
{
    sendMsgtoBlock(myblockNumber(), updatestatisticsMsg);
}

on endSim
{
    //Clean up lists
    ListDisposeAll(myBlockNumber());
}

on ContinueSim
{
    attribValuesIndex = GaGetIndex("_AttribValues");
}

```

B.5 Post Processing Scripts

Code and scripts in VB used to analyse the data output from ExtendSim.

```

Public Function simCT(dataArray() As Variant, steadyState As Double) As Double
    'gets the average simulation CT
    Dim i, count As Long
    Dim sum As Double

    sum = 0
    count = 0
    For i = 1 To UBound(dataArray)
        If dataArray(i, 1) = 1 Then
            If dataArray(i, 6) > steadyState Then
                sum = sum + (dataArray(i, 6) - dataArray(i, 3))
            
```

```

        count = count + 1
    End If
End If
Next i

simCT = sum / count
End Function

```

```

Public Function GetIntervals(dataArray() As Variant, timeInc As Double, numIncs As Integer) As
Variant
    'returns a 2 column array containing the warmup inc and the average ct value for that period
    Dim i, j, k, z As Long
    Dim tempArray(), workXArray(), valuesArray() As Variant
    Dim count, numExited, endRow, startRow As Long
    Dim startVal, endVal, aveCTPeriod As Double

    ReDim workXArray(numIncs)
    For i = 1 To UBound(workXArray)
        workXArray(i) = i * timeInc
    Next i

    'Find number of used rows
    count = 0
    For i = 1 To UBound(dataArray)
        If dataArray(i, 1) = 1 And dataArray(i, 6) > 0 Then 'lot did exit
            count = count + 1
        End If
    Next i

    ReDim tempArray(count, 2)
    'Col1 is the exit time, col2 is the CT
    count = 0
    For i = 1 To UBound(dataArray)
        If dataArray(i, 1) = 1 And dataArray(i, 6) > 0 Then 'lot did exit
            count = count + 1
            tempArray(count, 1) = dataArray(i, 6)
            tempArray(count, 2) = dataArray(i, 6) - dataArray(i, 3)
        End If
    Next i

    'Sort by the exit time (NB this was changed now the sorting is done in Extend, much faster)
    'tempArray = BubbleSortMultiCol(tempArray, 1)

    ReDim valuesArray(UBound(workXArray))
    j = 1
    For i = 1 To UBound(workXArray)
        If i = 1 Then
            startVal = 0
            endVal = workXArray(i)
        Else
            startVal = workXArray(i - 1)
            endVal = workXArray(i)
        End If

        Do Until tempArray(j, 1) >= startVal Or j >= UBound(tempArray)
            j = j + 1
        Loop

        k = 1
        Do Until tempArray(j + k, 1) > endVal Or (j + k) >= UBound(tempArray)
            k = k + 1
        Loop

        startRow = j
        endRow = j + k - 1

        numExited = (endRow - startRow) + 1
        aveCTPeriod = 0
        For z = startRow To endRow
            aveCTPeriod = aveCTPeriod + tempArray(z, 2)
        Next z

        aveCTPeriod = aveCTPeriod / numExited
        valuesArray(i) = aveCTPeriod
        j = endRow + 1
    Next i

    GetIntervals = valuesArray

```

End Function

```

Public Function simUTIL(dataArray() As Variant, simStartTime As Double, simEndTime As Double,
    Optional uOverA As Integer) As Double
    'gets the average tool utilisation for the simulation

    Dim i, count As Long
    Dim idleTime, timeInProgress, timeInRepair, totalRunTime As Double

    idleTime = 0
    timeInProgress = 0
    timeInRepair = 0

    count = 0
    For i = 1 To UBound(dataArray)
        If dataArray(i, 1) = 1 And dataArray(i, 3) > simStartTime And dataArray(i, 3) < simEndTime
            Then
                If dataArray(i, 6) = 0 Then
                    If dataArray(i, 4) <> 0 And dataArray(i, 5) <> 0 Then
                        idleTime = idleTime + dataArray(i, 4) - dataArray(i, 3)
                        timeInProgress = timeInProgress + dataArray(i, 5) - dataArray(i, 4)
                    End If
                ElseIf dataArray(i, 6) = 1 Then
                    If dataArray(i, 8) <> 0 And dataArray(i, 9) <> 0 Then
                        idleTime = idleTime + dataArray(i, 8) - dataArray(i, 3)
                        timeInRepair = timeInRepair + dataArray(i, 9) - dataArray(i, 8)
                    End If
                End If
            End If
        Next i

    If uOverA Then
        simUTIL = 1 - (idleTime / (idleTime + timeInProgress))
    Else
        simUTIL = 1 - (idleTime / (idleTime + timeInProgress + timeInRepair))
    End If

End Function

```

Semiconductor Wafer Manufacturing Data Format Specification

This chapter describes the format for the *Semiconductor Wafer Manufacturing Data Format Specification*, as outlined by Feigin et al. (1994) and Fowler and Robinson (1995) and made available to the public by SEMATECH at <http://www.eas.asu.edu/~masmlab/ftp.htm>. The format was formed to address the lack a factory level representative data available for academics and industrial engineers to experiment with product flows and compare fab specification. Currently there exists eight sample datasets (see Table C.7 on pg. C-9), some of which have been constructed by the authors of the format and others that have been donated by anonymous fabs and have been desensitised. The format consists of six files per dataset. The purpose of the files are listed in Table C.1.

Table C.1: Data files used for wafer data format specification.

File	Suffix ID	Description	Ref
Process Route	pr	Process route information for all processes	Table C.2
Rework Sequences	rw	Information on rework sequences	Table C.3
Tool Set	ts	Information on tools	Table C.4
Operator Set	os	Information on operators	Table C.5
Volume Release	vr	Release rate information	Table C.6
Comment File	cf	General comments and sample run results	n/a

C.1 File Description Overview

The volume release file divides product groups into specific recipes known as process flows and describes how the product is released into the factory. The process route file details the operations list (or steps) that each process flow follows. Each step contains processing information for a particular operation including batching and setup requirements, type of operator, toolset and processing pattern. It also contains post-processing information such as yield and rework probabilities and transport mechanisms. The rework sequence file is very similar to the process route file and contains all the processing and routing information for lots that must undergo a rework path. Once this rework path is complete the lots rejoining the ‘normal’ process route.

The operator set file contains information about the quantity of operators and their break requirements. Similarly the toolset file contains tool information such as tool quantity per toolset, wafer-based and time-based downtime patterns (maximum of five) and the percentage time required by operators for each phase of operation on the tool. The comments file contains examples of output from simulations and a narrative of any further general information about the factory that the dataset is based on.

C.2 File Descriptions

Tables C.2 to C.6 list the format for each of the file types listed in Table C.1. Details included are the field name, the data type (string, float, integer), the number of characters

reserved in the file for each field and some remarks about the field.

Table C.2: Structure of PROCESS ROUTE (*pr*) file.

<i>Field Name</i>	<i>Type</i>	<i>Width</i>	<i>Remarks</i>
Process Flow ID	String	10	Unique process flow ID
Step ID	String	10	Process step ID
Operation Description	String	25	A brief operation description
Tool Set ID	String	10	Assume 1 tool needed from this tool group for this operation
Operator Set ID	String	15	Assume 1 operator needed for this operation
Load Time	Float	10	Time to load wafers/lot/batch into tool
Unload Time	Float	10	Time to unload wafers/lot/batch from tool
Time per Wafer in Process	Float	10	Processing time per wafer (as appropriate)
Wafer Travel Time	Float	10	Travel time within tool (as appropriate)
Time per Lot	Float	10	Lot processing time (as appropriate)
Time per Batch	Float	10	Batch processing time (as appropriate)
Min Batch Size	Integer	5	Minimum batch size (in wafers)
Max Batch Size	Integer	5	Maximum batch size (in wafers)
Batch ID	String	10	Used to identify which operations can be batched together
Time per Spec. Setup	Float	10	Setup time required for changing from one process flow step (recipe spec.) to another
Time per Group Setup	Float	10	Setup time required for changing from one 'group' to another where group is specified by the Setup Group ID
Setup Group ID	String	15	See above
Lot Scrap Probability	Float	10	Prob. an entire lot is scrapped after this operation
Wafer Scrap Probability	Float	10	Prob. a wafer in a lot is scrapped after this operation
Lot Rework Probability	Float	10	Prob. an entire lot is sent for rework after this operation given it has not been scrapped
Wafer Rework Probability	Float	10	Prob. a wafer in a lot is sent for rework given it is has not been scrapped
Rework Sequence ID	String	10	Rework sequence to follow
Rework Return Step ID	String	10	Step to which wafers return after rework
Travel Time	Float	10	Travel time to next operation process step
Travel Time Operator ID	String	15	Operator set needed for travel

Table C.3: Structure of REWORK SEQUENCE (*rw*) file.

<i>Field Name</i>	<i>Type</i>	<i>Width</i>	<i>Remarks</i>
Rework Sequence ID	String	10	Unique process flow ID
Step ID	String	10	Process step ID
Operation Description	String	25	A brief operation description
Tool Set ID	String	10	Assume 1 tool needed from this tool group for this operation
Operator Set ID	String	15	Assume 1 operator needed for this operation
Load Time	Float	10	Time to load wafers/lot/batch into tool
Unload Time	Float	10	Time to unload wafers/lot/batch from tool
Time per Wafer in Process	Float	10	Processing time per wafer (as appropriate)
Wafer Travel Time	Float	10	Travel time within tool (as appropriate)
Time per Lot	Float	10	Lot processing time (as appropriate)
Time per Batch	Float	10	Batch processing time (as appropriate)
Min Batch Size	Integer	5	Minimum batch size (in wafers)
Max Batch Size	Integer	5	Maximum batch size (in wafers)
Batch ID	String	10	Used to identify which operations can be batched together
Time per Spec. Setup	Float	10	Setup time required for changing from one process flow step (recipe spec.) to another
Time per Group Setup	Float	10	Setup time required for changing from one 'group' to another where group is specified by the Setup Group ID
Setup Group ID	String	15	See above
Lot Scrap Probability	Float	10	Prob. an entire lot is scrapped after this operation
Wafer Scrap Probability	Float	10	Prob. a wafer in a lot is scrapped after this operation
Lot Rework Probability	Float	10	Prob. an entire lot is sent for rework after this operation given it has not been scrapped
Wafer Rework Probability	Float	10	Prob. a wafer in a lot is sent for rework given it is has not been scrapped
Rework Sequence ID	String	10	Rework sequence to follow
Rework Return Step ID	String	10	Step to which wafers return after rework
Travel Time	Float	10	Travel time to next operation process step
Travel Time Operator ID	String	15	Operator set needed for travel

Table C.4: Structure of TOOL SET (*ts*) file.

<i>Field Name</i>	<i>Type</i>	<i>Width</i>	<i>Remarks</i>
Tool Set ID	String	10	Tool set identifier
Tool Description	String	25	Name or description of tool
Quantity	Integer	5	Number of (identical) tools in tool set
Operator Load Fraction	Float	10	Fraction of time operator is needed for lot loading
Operator Unload Fraction	Float	10	Fraction of time operator is needed for lot unloading
Operator Process Fraction	Float	10	Fraction of time operator is needed for lot processing
Down Time #1 Description	String	25	Description of tool down time type 1
Down Time #1 Type	Integer	5	0 = time based; 1 = run based
Time or Runs Between #1	Float	15	Mean time (runs) between this down time event
Duration #1	Float	10	Duration of this down time event
Down Time #1 Operator Set ID	String	15	Operator needed from this set during this down time event
Down Time #2 Description	String	25	Description of tool down time type 2
Down Time #2 Type	Integer	5	0 = time based; 1 = run based
Time or Runs Between #2	Float	15	Mean time (runs) between this down time event
Duration #2	Float	10	Duration of this down time event
Down Time #2 Operator Set ID	String	15	Operator needed from this set during this down time event
Down Time #3 Description	String	25	Description of tool down time type 3
Down Time #3 Type	Integer	5	0 = time based; 1 = run based
Time or Runs Between #3	Float	15	Mean time (runs) between this down time event
Duration #3	Float	10	Duration of this down time event
Down Time #3 Operator Set ID	String	15	Operator needed from this set during this down time event
Down Time #4 Description	String	25	Description of tool down time type 4
Down Time #4 Type	Integer	5	0 = time based; 1 = run based
Time or Runs Between #4	Float	15	Mean time (runs) between events
Duration #4	Float	10	Duration of this down time event
Down Time #4 Operator Set ID	String	15	Operator needed from this set during this down time event
Down Time #5 Description	String	25	Description of tool down time type 5
Down Time #5 Type	Integer	5	0 = time based; 1 = run based
Time or Runs Between #5	Float	15	Mean time (runs) between events
Duration #5	Float	10	Duration of this down time event
Down Time #5 Operator Set ID	String	15	Operator needed from this set during this down time even

Table C.5: Structure of OPERATOR SET (*os*) file.

<i>Field Name</i>	<i>Type</i>	<i>Width</i>	<i>Remarks</i>
Operator Set ID	String	15	Operator set identifier
Operator Description	String	25	Operator Set Name
Quantity	Integer	5	Number of operators in this set
Break #1 Description	String	25	Description of break type
Time Between #1	Float	15	Time between breaks of this type
Duration #1	Float	10	Duration of breaks of this type
Break #2 Description	String	25	Description of break type
Time Between #2	Float	15	Time between breaks of this type
Duration #2	Float	10	Duration of breaks of this type
Break #3 Description	String	25	Description of break type
Time Between #3	Float	15	Time between breaks of this type
Duration #3	Float	10	Duration of breaks of this type

Table C.6: Structure of VOLUME RELEASE (*vr*) file.

<i>Field Name</i>	<i>Type</i>	<i>Width</i>	<i>Remarks</i>
Process Flow	String	10	Process Flow ID
Product ID	String	10	Unique product family ID
Product Name	String	25	Name of Product (optional)
Start Rate	Float	10	Number of wafers per day released into line based on 7 day/week operation
Lot Size	Integer	10	Number of wafers in a released lot

C.3 Additional Information

This section describes important information about how the data sets should be interpreted and how the information that they contain should be implemented in a simulation. Finally, Table C.7 lists some descriptive attributes regarding each of the datasets.

1. All times are specified in minutes except if stated explicitly otherwise.
2. All tools in a tool set and all operators in an operator set are considered identical. In particular, all tools in a tool set are considered qualified to perform all operations coming to that tool set. The same holds true for operators.
3. An operation is uniquely specified by its Process ID (or Rework ID) and Step ID. However, we allow multiple entries for the same Product ID, Step ID pair. The resolution of such multiple entries should be made in the description field. Multiple entries might be used to specify alternative tool sets or operator sets that can be used to perform an operation.
4. The following formulas can be used to calculate processing time per lot (pt), time until tool becomes free (tf), and total lot cycle time through an operation (ct):

$$\begin{aligned} \text{pt} &= \text{Time per Batch} * \text{No. of batches required for the lot} \\ &\quad + \text{Time per Lot} \\ &\quad + \text{Time per Wafer in Process} * \text{No. of wafers in the lot} \\ &\quad + \text{Product Setup (if appropriate)} \\ &\quad + \text{Group Setup (if appropriate)} \\ \text{tf} &= \text{Load Time} + \text{pt} + \text{Unload Time} \\ \text{ct} &= \text{Load Time} + \text{pt} + \text{Wafer Travel Time} + \text{Unload Time} \end{aligned}$$

5. No distributional information is included in the data sets beyond first moment (mean) information. Fields with names like Load Time have the implicit prefix qualifier Mean.
6. The wafer start rates given in the Volume/release file are intended as guidelines. The exact method by which lots are released is to be determined by the users of the data.
7. Information on process holds, engineering holds and send aheads is not included.

8. Time bound sequences cannot be specified explicitly.
9. The following types of tools typically found in a semiconductor wafer manufacturing line can be modelled by making appropriate use of the processing time parameters:
 - (a) Single wafer tools. Set `Time per Wafer in Process` field to wafer processing time and all other process time parameters to zero.
 - (b) Batch tools. Set `Min Batch Size` and `Max Batch Size` fields appropriately. Set `Time per Batch` field to batch processing time and all other process time parameters to zero. Lots with the same batch ID may be batched together (up to the maximum batch size). Where batch ID's are left blank only lots at the same step of the same process flow may be batched together.
 - (c) Multi-Sequence tools. Set `Min Batch Size` and `Max Batch Size` fields appropriately. Set `Time per Batch` field to the largest single tank time and set `Wafer Travel Time` field to the remaining tank time. Set other process time parameters to zero.
 - (d) Conveyor tools. Set `Time per Wafer in Process` field to wafer processing time and `Wafer Travel Time` field to the wafer travel time. Set other process time parameters to zero.
 - (e) Inspect tools. Set `Time per Lot` field to inspect time. Set other process time parameters to zero.
 - (f) Certain cluster tools. Model as batch tools.
 - (g) Linked track/linked lithography tools. Model as conveyor tools.
10. Group and spec. setups are assumed to be done whenever the group or process flow step ID changes from one operation to another. However, the duration of the setup does not vary as a function of the specific group or product preceding the operation.
11. An `Operator Set ID` or `Tool Set ID` field that is blank indicates that no resource is needed for that step.

Table C.7: Comparison of Sematech datasets.

Type of product	minifab	Datasets						
		set 1	set 2	set 3	set 4	set 5	set 6	set 7
	wafers	non-volatile memory	ASIC & memory	memory	micro-processors	ASIC	ASICS	R&D wafer
Number of process flows	1	2	7	11	2	21	9	1
Number of products	3	2	7	11	7	177	9	1
Dataset makes up what % of factory	n/a	95-98%	undisc.	86%	100%	98%	95-98%	100%
Ave. number of process steps per mask layer	n/a	15	26	35	10	30	30	14
Operators included	yes	yes	yes	no	no	yes	yes	no
Rework included	no	yes	yes	yes	no	no	no	no
Equipment setups included	no	yes	yes	no	no	no	no	no
Scrap and yield included	no	yes	yes	yes	no	no	yes	no
Equipment downtime included	yes	yes	yes	yes	yes	yes	yes	yes
Number of equipment groups	3	83	97	73	35	85	104	24
Approximate wafer starts per month	9,000	30,000	10,000	21,400	3,400	10,000	5,500	408

Code for Fab Model A

This chapter includes the code used to create the VB implementation of the Sematech full fab simulation. A description of the application is given in Chapter 5.

D.1 Simulation Model Inputs

The simulation model inputs are taken from local dataset text files, cleaned up and input to ExtendSim's database for easier access, speed and reliability during model runtime.

Table D.1 lists the subroutines and functions used in this process.

```
Sub GetInputData()  
    'There are 6 files that contain information about the fab cf, ts, os, pr, rw, vr  
    'cf contains general text information about the fab (unused)  
    'ts contains toolset information  
    'os contains operator information  
    'pr contains process route information  
    'rw contains rework sequence information  
    'vr contains volume release information  
  
    Dim sourceFile As String, targetFile As String  
    Dim DBName As String  
    Dim FieldNames() As Variant  
    Dim FieldTypes() As Variant
```

Table D.1: Input data subroutines and functions for the Sematech model.

Subroutine	Purpose
<i>GetInputData()</i>	Pulls the Sematech dataset text files from the local sources.
<i>CleanUpVRData()</i>	Formats the information from the <i>volume release</i> file.
<i>CleanUpOsData()</i>	Formats the information from the <i>operator set</i> file.
<i>CleanUpTSData()</i>	Formats the information from the <i>toolset</i> file.
<i>CleanUpPRData()</i>	Formats the information from the <i>process route</i> file.
<i>CleanUpRWData()</i>	Formats the information from the <i>rework sequence</i> file.
<i>GetFieldNames</i>	Function that returns the field names of the datasets.
<i>GetFieldTypes</i>	Function that returns the field types (string, value, integer) of the field names.

```

Dim dbIndex As Integer
Dim fieldIndex As Integer
Dim tableIndex As Integer
Dim tempFolderPath As String
Dim i, j As Integer
Dim dataArray() As Variant
Dim tableExist As Boolean

tableExist = False
DBName = "SematechDB"
dbIndex = CreateExtendSimDB(DBName, True) 'create and overwrite if necessary

'create temp folder in dataset location
tempFolderPath = datasetLocation & "\" & "temp"
CreateAFolder tempFolderPath

For i = 1 To 5
    sourceFile = datasetLocation & "\" & datasetName & "." & dataTable(i)
    targetFile = tempFolderPath & "\" & dataTable(i) & ".txt"
    If FileOrDirExists(sourceFile) Then
        Select Case dataTable(i)
            Case "vr"
                vrData = ReadSematechFileToArray(sourceFile)
                If Not (IsEmpty(vrData)) Then
                    CleanUpVRData
                    writeArrayToCSV vrData, targetFile
                    tableExist = True
                End If
            Case "os"
                osData = ReadSematechFileToArray(sourceFile)
                If Not (IsEmpty(osData)) Then
                    CleanUpOsData
                    writeArrayToCSV osData, targetFile
                    tableExist = True
                End If
            Case "ts"
                tsData = ReadSematechFileToArray(sourceFile)
                If Not (IsEmpty(tsData)) Then
                    CleanUpTSData
                    writeArrayToCSV tsData, targetFile
                    tableExist = True
                End If
            Case "pr"
                prData = ReadSematechFileToArray(sourceFile)
                If Not (IsEmpty(prData)) Then
                    CleanUpPRData
                    writeArrayToCSV prData, targetFile
                    tableExist = True
                End If
            Case "rw"
                rwData = ReadSematechFileToArray(sourceFile)
                If Not (IsEmpty(rwData)) Then
                    CleanUpRWData
                    writeArrayToCSV rwData, targetFile
                    tableExist = True
                End If
        End Select

        If tableExist Then
            FieldNames = GetFieldNames(dataTable(i))
            FieldTypes = GetFieldTypes(dataTable(i))
        End If
    End If
Next i

```

Appendix D. Code for Fab Model A

```
        tableIndex = CreateExtendSimTable(DBName, dataTable(i), True)
    For j = 1 To UBound(FieldNames)
        fieldIndex = CreateExtendSimField(DBName, dataTable(i), FieldNames(j), FieldTypes(j)
            , 2, True)
    Next j
    ExtendDBTableImportText DBName, dataTable(i), targetFile
    tableExist = False
End If
End If
Next i
End Sub
```

```
Function GetFieldNames(dataTableType As String) As Variant
    'Returns a single column array containing a list of the field names for the dataset file

    Dim infoArray() As Variant
    Dim ext As String
    Dim ws As Worksheet
    Dim y As Integer, i As Integer

    For Each ws In ThisWorkbook.Worksheets
        If Mid(ws.name, 1, 2) = dataTableType Then
            Exit For
        End If
    Next ws

    y = ws.Cells(1, 1).End(xlDown).Row
    ReDim infoArray(y)
    For i = 1 To y
        infoArray(i) = CStr(ws.Cells(i, 1))
    Next i

    GetFieldNames = infoArray
End Function
```

```
Function GetFieldTypes(dataTableType As String) As Variant
    'Returns a single column array containing a list of the field types for the dataset file
    'Integer is 4096
    'Float is 8192
    'String is 16384

    Dim infoArray() As Variant
    Dim ext As String
    Dim ws As Worksheet
    Dim y As Integer, i As Integer

    For Each ws In ThisWorkbook.Worksheets
        If Mid(ws.name, 1, 2) = dataTableType Then
            Exit For
        End If
    Next ws

    y = ws.Cells(1, 1).End(xlDown).Row
    ReDim infoArray(y)
    For i = 1 To y
        Select Case ws.Cells(i, 2)
            Case "Integer"
                infoArray(i) = 4096
            Case "Float"
                infoArray(i) = 8192
            Case "String"
                infoArray(i) = 16384
            Case Else
                MsgBox ("Spurious data type in worksheet " & ws.name & ". Operation aborted")
                GetFieldTypes = Empty
                Exit Function
        End Select
    Next i

    GetFieldTypes = infoArray
End Function
```

```

Public Sub CleanUpVRData()
    'Process Flow      -change to Integer store unique string values in ProcessFlow
    'Product ID       -change to integer, store unique string values in ProductID
    'Product Name     -unchanged, store in ProductName corresponding to ProductID
    'Start Rate
    'Lot Size

    Dim i As Integer, y As Integer, x As Integer, j As Integer, y2 As Integer
    Dim tempArray() As Variant

    y = UBound(vrData)
    ReDim tempArray(y)

    For i = 1 To y
        tempArray(i) = vrData(i, 1)
    Next i

    ProcessFlow = UniqueItems(tempArray, False)
    y2 = UBound(ProcessFlow)

    For i = 1 To y
        For j = 1 To y2
            If vrData(i, 1) = ProcessFlow(j) Then
                vrData(i, 1) = Int(j)
            Exit For
        End If
    Next j
    Next i

    ReDim ProductName(y)
    ReDim ProductID(y)
    For i = 1 To y
        ProductID(i) = vrData(i, 2)
        ProductName(i) = vrData(i, 3)
        vrData(i, 2) = i
    Next i

End Sub

```

```

Sub CleanUpOsData()
    'Operator Set ID          -store unique values in OperatorSetID
    'Operator Description    -store unique values in OperatorDescription
    'Quantity
    'Break #1 Description
    'Time Between #1        -change to hour basis if hourflag is true
    'Duration #1            -change to hour basis if hourflag is true
    'Break #2 Description
    'Time Between #2        -change to hour basis if hourflag is true
    'Duration #2            -change to hour basis if hourflag is true
    'Break #3 Description
    'Time Between #3        -change to hour basis if hourflag is true
    'Duration #3            -change to hour basis if hourflag is true

    Dim i As Integer, y As Integer, x As Integer, j As Integer, y2 As Integer
    Dim tempArray() As Variant

    y = UBound(osData)
    ReDim tempArray(y)

    For i = 1 To y
        tempArray(i) = osData(i, 1)
    Next i

    OperatorSetID = UniqueItems(tempArray, False)
    y2 = UBound(OperatorSetID)
    ReDim OperatorDescription(y2)

    For i = 1 To y
        For j = 1 To y2
            If osData(i, 1) = OperatorSetID(j) Then
                osData(i, 1) = Int(j)
                OperatorDescription(j) = osData(i, 2)
            Exit For
        End If
    Next j
    Next i

    If hourFlag Then

```


Appendix D. Code for Fab Model A

```

For i = 1 To y
    osData(i, 5) = osData(i, 5) / 60
    osData(i, 6) = osData(i, 6) / 60
    osData(i, 8) = osData(i, 8) / 60
    osData(i, 9) = osData(i, 9) / 60
    osData(i, 11) = osData(i, 11) / 60
    osData(i, 12) = osData(i, 12) / 60
Next i
End If
End Sub

```

```

Sub CleanUpTSData()
    'Toolset ID -change to integer, store unique values in ToolsetID
    'Tool Description -store unique values in ToolDescription
    'Quantity
    'Operator Load Fraction
    'Operator Unload Fraction
    'Operator Process Fraction
    'Downtime #1 Description
    'Downtime #1 Type
    'Time or Runs Between #1 -change to hour if hourFlag is true and the DT type is 0
    'Duration #1 -change to hour if hourFlag is true
    'Downtime #1 Operator Set ID -change to integer
    'Downtime #2 Description
    'Downtime #2 Type
    'Time or Runs Between #2 -change to hour if hourFlag is true and the DT type is 0
    'Duration #2 -change to hour if hourFlag is true
    'Downtime #2 Operator Set ID -change to integer
    'Downtime #3 Description
    'Downtime #3 Type
    'Time or Runs Between #3 -change to hour if hourFlag is true and the DT type is 0
    'Duration #3 -change to hour if hourFlag is true
    'Downtime #3 Operator Set ID -change to integer
    'Downtime #4 Description
    'Downtime #4 Type
    'Time or Runs Between #4 -change to hour if hourFlag is true and the DT type is 0
    'Duration #4 -change to hour if hourFlag is true
    'Downtime #4 Operator Set ID -change to integer
    'Downtime #5 Description
    'Downtime #5 Type
    'Time or Runs Between #5 -change to hour if hourFlag is true and the DT type is 0
    'Duration #5 -change to hour if hourFlag is true
    'Downtime #5 Operator Set ID -change to integer

    Dim i As Integer, y As Integer, x As Integer, j As Integer, y2 As Integer
    Dim tempArray() As Variant

    y = UBound(tsData)
    ReDim tempArray(y)

    For i = 1 To y
        tempArray(i) = tsData(i, 1)
    Next i

    ToolSetID = UniqueItems(tempArray, False)
    y2 = UBound(ToolSetID)
    ReDim ToolDescription(y2)
    ReDim ToolQuantity(y2)

    For i = 1 To y
        For j = 1 To y2
            If tsData(i, 1) = ToolSetID(j) Then
                tsData(i, 1) = Int(j)
                ToolQuantity(i) = tsData(i, 3)
                ToolDescription(j) = tsData(i, 2)
            Exit For
        End If
    Next j
    Next i

    If Not (IsEmpty(osData)) Then
        For i = 1 To y
            For j = 1 To UBound(OperatorSetID)
                If OperatorSetID(j) = tsData(i, 11) Then
                    tsData(i, 11) = j
                Exit For
            End If
        Next j
    End If

```

```

For j = 1 To UBound(OperatorSetID)
  If OperatorSetID(j) = tsData(i, 16) Then
    tsData(i, 16) = j
  Exit For
  End If
Next j

For j = 1 To UBound(OperatorSetID)
  If OperatorSetID(j) = tsData(i, 21) Then
    tsData(i, 21) = j
  Exit For
  End If
Next j

For j = 1 To UBound(OperatorSetID)
  If OperatorSetID(j) = tsData(i, 26) Then
    tsData(i, 26) = j
  Exit For
  End If
Next j

For j = 1 To UBound(OperatorSetID)
  If OperatorSetID(j) = tsData(i, 31) Then
    tsData(i, 31) = j
  Exit For
  End If
Next j
Next i
End If

If hourFlag Then
  For i = 1 To y
    If tsData(i, 8) = 0 Then
      tsData(i, 9) = tsData(i, 9) / 60
      tsData(i, 10) = tsData(i, 10) / 60
    Else
      tsData(i, 10) = tsData(i, 10) / 60
    End If

    If tsData(i, 13) = 0 Then
      tsData(i, 14) = tsData(i, 14) / 60
      tsData(i, 15) = tsData(i, 15) / 60
    Else
      tsData(i, 15) = tsData(i, 15) / 60
    End If

    If tsData(i, 18) = 0 Then
      tsData(i, 19) = tsData(i, 19) / 60
      tsData(i, 20) = tsData(i, 20) / 60
    Else
      tsData(i, 20) = tsData(i, 20) / 60
    End If

    If tsData(i, 23) = 0 Then
      tsData(i, 24) = tsData(i, 24) / 60
      tsData(i, 25) = tsData(i, 25) / 60
    Else
      tsData(i, 25) = tsData(i, 25) / 60
    End If

    If tsData(i, 28) = 0 Then
      tsData(i, 29) = tsData(i, 29) / 60
      tsData(i, 30) = tsData(i, 30) / 60
    Else
      tsData(i, 30) = tsData(i, 30) / 60
    End If
  Next i
End If
End Sub

```

```

Sub CleanUpPRData()
  'Process Flow ID -change to integer
  'Step ID -change to integer steps of 1 for each
  'Operation Description -store in OperationDescription
  'Tool Set ID -change to integer, from ToolsetID array
  'Operator Set ID -change to integer, from OperatorSetID array
  'Load Time -change to hour if hourflag is true

```

```

'Unload Time -change to hour if hourflag is true
'Time Per Wafer in Process -change to hour if hourflag is true
'Wafer Travel Time -change to hour if hourflag is true
'Time per Lot -change to hour if hourflag is true
'Time Per Batch -change to hour if hourflag is true
'Min Batch Size
'Max Batch Size
'Batch ID -change to integer, store unique entries in BatchID
'Time per Spec. Setup -change to hour if hourflag is true
'Time per Group Setup -change to hour if hourflag is true
'Setup Group ID -change to integer, store unique entries in setupGroupID
'Lot scrap Probability
'Wafer Scrap Probability
'Lot Rework Probability
'Wafer Rework Probability
'Rework Sequence ID
'Rework Return Step ID
'Travel Time -change to hour if hourflag is true
'Travel Time Operator -change to integer from operatorSetID

Dim i As Integer, y As Integer, x As Integer, j As Integer, y2 As Integer, k As Integer
Dim tempArray() As Variant
Dim count As Integer
Dim counter() As Integer
Dim tempUpperBound As Integer
Dim inc As Integer
Dim findStep As Variant
Dim findProcess As Integer

y = UBound(prData)
ReDim counter(UBound(ProcessFlow))

For k = 1 To UBound(counter)
    counter(k) = 0
Next k

For i = 1 To y
    For j = 1 To UBound(ProcessFlow)
        If prData(i, 1) = ProcessFlow(j) Then
            prData(i, 1) = CInt(j)
            counter(j) = counter(j) + 1
        Exit For
    End If
Next j
Next i

tempUpperBound = WorksheetFunction.Max(counter)

For k = 1 To UBound(counter)
    counter(k) = 0
Next k

ReDim StepID(tempUpperBound, UBound(ProcessFlow))
ReDim OperationDescription(tempUpperBound, UBound(ProcessFlow))

For i = 1 To UBound(StepID)
    For j = 1 To UBound(StepID, 2)
        StepID(i, j) = 0
        OperationDescription(i, j) = ""
    Next j
Next i

For i = 1 To y
    inc = prData(i, 1)
    counter(inc) = counter(inc) + 1
    StepID(counter(inc), inc) = prData(i, 2)
    OperationDescription(counter(inc), inc) = prData(i, 3)
    prData(i, 2) = counter(inc)
Next i

'Rename tools
For i = 1 To y
    For j = 1 To UBound(ToolSetID)
        If ToolSetID(j) = prData(i, 4) Then
            prData(i, 4) = j
        Exit For
    End If
Next j
Next i

'Get BatchID
count = 0
For i = 1 To y

```

```

If Not Len(prData(i, 14)) = 0 Then
    count = count + 1
    ReDim Preserve tempArray(count)
    tempArray(count) = prData(i, 14)
End If
Next i

If count > 0 Then
    BatchID = UniqueItems(tempArray, False)
    For i = 1 To y
        For j = 1 To UBound(BatchID)
            If Not Len(prData(i, 14)) = 0 Then
                If BatchID(j) = prData(i, 14) Then
                    prData(i, 14) = CInt(j)
                    Exit For
                End If
            End If
        Next j
    Next i
End If

'Now check if a batchID should be assigned(for lots that have no batch Id but are batched by
'their StepID and ProcessFlowID)
If count > 0 Then
    count = UBound(BatchID)
Else
    count = 0
End If

For i = 1 To y
    If (Len(prData(i, 14)) = 0) And (prData(i, 12) > 0) And (prData(i, 13) > 0) Then
        count = count + 1
        ReDim Preserve BatchID(count)
        BatchID(count) = "_" & prData(i, 1) & "0000" & prData(i, 2)
        prData(i, 14) = count
    End If
Next i

'Get SetupGroupID
count = 0
For i = 1 To y
    If Not Len(prData(i, 17)) = 0 Then
        count = count + 1
        ReDim Preserve tempArray(count)
        tempArray(count) = prData(i, 17)
    End If
Next i

If count > 0 Then
    SetupGroupID = UniqueItems(tempArray, False)
    For i = 1 To y
        For j = 1 To UBound(SetupGroupID)
            If Not Len(prData(i, 17)) = 0 Then
                If SetupGroupID(j) = prData(i, 17) Then
                    prData(i, 17) = CInt(j)
                    Exit For
                End If
            End If
        Next j
    Next i
End If

If Not (IsEmpty(osData)) Then
    'Rename operators
    For i = 1 To y
        For j = 1 To UBound(OperatorSetID)
            If OperatorSetID(j) = prData(i, 5) Then
                prData(i, 5) = j
                Exit For
            End If
        Next j
    Next i

    'Travel Operator
    For i = 1 To y
        For j = 1 To UBound(OperatorSetID)
            If OperatorSetID(j) = prData(i, 25) Then
                prData(i, 25) = j
                Exit For
            End If
        Next j
    Next i
End If

```

```

'Get Rework Return StepID
For i = 1 To y
  If Not Len(prData(i, 23)) = 0 Then
    findStep = prData(i, 23)
    findProcess = prData(i, 1)
    For j = 1 To UBound(StepID)
      If findStep = StepID(j, findProcess) Then
        prData(i, 23) = CInt(j)
        Exit For
      End If
    Next j
  End If
Next i

'Get ReworkSequenceID
count = 0
For i = 1 To y
  If Not Len(prData(i, 22)) = 0 Then
    count = count + 1
    ReDim Preserve tempArray(count)
    tempArray(count) = prData(i, 22)
  End If
Next i

If count > 0 Then
  ReworkSequenceID = UniqueItems(tempArray, False)
  For i = 1 To y
    If Not (Len(prData(i, 22)) = 0) Then
      For j = 1 To UBound(ReworkSequenceID)
        If ReworkSequenceID(j) = prData(i, 22) Then
          prData(i, 22) = j
          Exit For
        End If
      Next j
    End If
  Next i
End If

'Convert to hour
If hourFlag Then
  For i = 1 To y
    prData(i, 6) = CDec(prData(i, 6) / 60)
    prData(i, 7) = CDec(prData(i, 7) / 60)
    prData(i, 8) = CDec(prData(i, 8) / 60)
    prData(i, 9) = CDec(prData(i, 9) / 60)
    prData(i, 10) = CDec(prData(i, 10) / 60)
    prData(i, 11) = CDec(prData(i, 11) / 60)
    prData(i, 15) = CDec(prData(i, 15) / 60)
    prData(i, 16) = CDec(prData(i, 16) / 60)
    prData(i, 24) = CDec(prData(i, 24) / 60)
  Next i
End If
End Sub

```

```

Sub CleanUpRWData()
  'Rework Sequence ID -change to integer, store in ReworkSequenceID
  'Step ID -change to integer
  'Operation Description -store in OperationDescription
  'Tool Set ID -change to integer, from ToolsetID array
  'Operator Set ID -change to integer, from OperatorSetID array
  'Load Time -change to hour if hourflag is true
  'Unload Time -change to hour if hourflag is true
  'Time Per Wafer in Process -change to hour if hourflag is true
  'Wafer Travel Time -change to hour if hourflag is true
  'Time per Lot -change to hour if hourflag is true
  'Time Per Batch -change to hour if hourflag is true
  'Min Batch Size
  'Max Batch Size
  'Batch ID -change to integer, store unique entries in BatchID
  'Time per Spec. Setup -change to hour if hourflag is true
  'Time per Group Setup -change to hour if hourflag is true
  'Setup Group ID -change to integer from operatorsetID
  'Lot scrap Probability
  'Wafer Scrap Probability
  'Lot Rework Probability
  'Wafer Rework Probability
  'Rework Sequence ID -ignore (typically unused)

```

```

'Rework Return Step ID -ignore(ignore typically unused)
'Travel Time -change to hour if hourflag is true
'Travel Time Operator -change to integer from operatorSetID

Dim i As Integer, y As Integer, x As Integer
Dim j As Integer, y2 As Integer, k As Integer
Dim tempArray() As Variant
Dim count As Integer, counter() As Integer, maxCount As Integer, inc As Integer
Dim BatchIDFound As Boolean, SetupGroupIDFound As Boolean

y = UBound(rwData)

For i = 1 To y
    For j = 1 To UBound(ReworkSequenceID)
        If ReworkSequenceID(j) = rwData(i, 1) Then
            rwData(i, 1) = j
            Exit For
        End If
    Next j
Next i

ReDim counter(UBound(ReworkSequenceID))
For k = 1 To UBound(counter)
    counter(k) = 0
Next k

'get Rework Step Id counts
For i = 1 To y
    For j = 1 To UBound(ReworkSequenceID)
        If rwData(i, 1) = j Then
            counter(j) = counter(j) + 1
            Exit For
        End If
    Next j
Next i

maxCount = WorksheetFunction.Max(counter)
ReDim ReworkStepID(maxCount, UBound(ReworkSequenceID))
ReDim ReworkDescription(maxCount, UBound(ReworkSequenceID))

For i = 1 To UBound(ReworkStepID)
    For j = 1 To UBound(ReworkStepID, 2)
        ReworkStepID(i, j) = 0
        ReworkDescription(i, j) = ""
    Next j
Next i

For k = 1 To UBound(counter)
    counter(k) = 0
Next k

For i = 1 To y
    inc = rwData(i, 1)
    counter(inc) = counter(inc) + 1
    ReworkStepID(counter(inc), inc) = rwData(i, 2)
    ReworkDescription(counter(inc), inc) = rwData(i, 3)
    rwData(i, 2) = counter(inc)
Next i

'Rename tools
For i = 1 To y
    For j = 1 To UBound(ToolSetID)
        If ToolSetID(j) = rwData(i, 4) Then
            rwData(i, 4) = j
            Exit For
        End If
    Next j
Next i

'Get BatchID
If Not (IsEmpty(BatchID)) Then
    count = UBound(BatchID)
Else
    count = 0
End If

For i = 1 To y
    If Not (Len(rwData(i, 14)) = 0) Then
        BatchIDFound = False
        If Not (IsEmpty(BatchID)) Then
            For j = 1 To UBound(BatchID)
                If BatchID(j) = rwData(i, 14) Then
                    rwData(i, 14) = j
                End If
            Next j
        End If
    End If
Next i

```

```

        BatchIDFound = True
    Exit For
End If
Next j
End If
If Not (BatchIDFound) Then
    count = count + 1
    ReDim Preserve BatchID(count)
    BatchID(count) = rwData(i, 14)
    rwData(i, 14) = count
End If
End If
Next i

'Now check if a batchID should be assigned(for lots that have no batch Id but are batched by
'their RWStepID and RWSequenceID). Note that these batchID are different scheme so we do not
search batchID
'array for them because we are referring to RWStepId and ReworkSequenceId.

If Not (IsEmpty(BatchID)) Then
    count = UBound(BatchID)
Else
    count = 0
End If

For i = 1 To y
    If (Len(rwData(i, 14)) = 0) And (rwData(i, 12) > 0) And (rwData(i, 13) > 0) Then
        count = count + 1
        ReDim Preserve BatchID(count)
        BatchID(count) = rwData(i, 1) & "0000" & rwData(i, 2)
        rwData(i, 14) = count
    End If
Next i

'Get SetupGroupID
If Not (IsEmpty(SetupGroupID)) Then
    count = UBound(SetupGroupID)
Else
    count = 0
End If

For i = 1 To y
    If Not (Len(rwData(i, 17)) = 0) Then
        SetupGroupIDFound = False
        If Not (IsEmpty(SetupGroupID)) Then
            For j = 1 To UBound(SetupGroupID)
                If SetupGroupID(j) = rwData(i, 17) Then
                    rwData(i, 17) = j
                    SetupGroupIDFound = True
                    Exit For
                End If
            Next j
        End If
        If Not (SetupGroupIDFound) Then
            count = count + 1
            ReDim Preserve SetupGroupID(count)
            SetupGroupID(count) = rwData(i, 17)
            rwData(i, 17) = count
        End If
    End If
Next i

If Not (IsEmpty(osData)) Then
    'Rename operators
    For i = 1 To y
        If Not (Len(rwData(i, 5)) = 0) Then
            For j = 1 To UBound(OperatorSetID)
                If OperatorSetID(j) = rwData(i, 5) Then
                    rwData(i, 5) = j
                    Exit For
                End If
            Next j
        End If
    Next i

    'Travel Operator
    For i = 1 To y
        If Not (Len(rwData(i, 25)) = 0) Then
            For j = 1 To UBound(OperatorSetID)
                If OperatorSetID(j) = rwData(i, 25) Then
                    rwData(i, 25) = j
                    Exit For
                End If
            Next j
        End If
    Next i

```

```
        Next j
      End If
    Next i
  End If

  'Convert to hour
  If hourFlag Then
    For i = 1 To y
      rwData(i, 6) = rwData(i, 6) / 60
      rwData(i, 7) = rwData(i, 7) / 60
      rwData(i, 8) = rwData(i, 8) / 60
      rwData(i, 9) = rwData(i, 9) / 60
      rwData(i, 10) = rwData(i, 10) / 60
      rwData(i, 11) = rwData(i, 11) / 60
      rwData(i, 15) = rwData(i, 15) / 60
      rwData(i, 16) = rwData(i, 16) / 60
      rwData(i, 24) = rwData(i, 24) / 60
    Next i
  End If
End Sub
```

Code for Fab Model B

This chapter includes the code used to create the Python/SimPy discrete event simulation (DES) model used in the Sematech fab model. A description of the application is given in Chapter 6.

```
# Declare Imports
from SimPy.Simulation import *
from SimPy.SimGUI import *
from SimPy.SimPlot import *
from math import ceil
from random import expovariate
from datetime import datetime
import random
import pylab
import os

# Import Data
def importData():
    "imports the data from Sematech data files"

    global vrdata # volume release list
    global prdata # process route list
    global osdata # operator set list
    global tsdata # toolset list
    global rwdata # rework list

    vrdata=[] # volume release list
    prdata=[] # process route list
    osdata=[] # operator set list
    tsdata=[] # toolset list
    rwdata=[] # rework list
```

```

if gui.params.Dataset==0:
    folderName='minifab'
    typeName='mf'
elif gui.params.Dataset==1:
    folderName='set1'
    typeName='set1'
elif gui.params.Dataset==2:
    folderName='set2'
    typeName='set2'
elif gui.params.Dataset==3:
    folderName='set3'
    typeName='set3'
elif gui.params.Dataset==4:
    folderName='set4'
    typeName='set4'
elif gui.params.Dataset==5:
    folderName='set5'
    typeName='set5'
elif gui.params.Dataset==6:
    folderName='set6'
    typeName='set6'
elif gui.params.Dataset==7:
    folderName='set7'
    typeName='set7'

dataPath = 'C:/Documents and Settings/nbyrn3x/My Documents/Sematech Datasets'
dataFileExt = ['vr', 'pr', 'os', 'ts', 'rw']

for i in range(len(dataFileExt)):
    if os.path.exists(dataPath + '/' + folderName + '/' + typeName + '.' + dataFileExt[i]):
        file=open(dataPath + '/' + folderName + '/' + typeName + '.' + dataFileExt[i], 'r')
        for line in file.readlines():
            temp=[]
            for elmt in line.split(','):
                el=elmt.rstrip('\n')
                temp.append(el)

            if dataFileExt[i] == 'vr':
                vrdata.append(temp)
            elif dataFileExt[i] == 'pr':
                prdata.append(temp)
            elif dataFileExt[i] == 'os':
                osdata.append(temp)
            elif dataFileExt[i] == 'ts':
                tsdata.append(temp)
            elif dataFileExt[i] == 'rw':
                rwdata.append(temp)
        else:
            "The file " + dataPath + '/' + folderName + '/' + \
            typeName + '.' + dataFileExt[i] + " could not be found."

# Misc Functions

def stringToValChecker(item):
    """check a string to see if it is
    not empty and greater than zero """
    valid = False
    if len(item) <> 0:
        if float(item) > 0:
            valid = True
    return valid

class timer(Process):
    "timer function estimates the remining run time"

    def start(self, simulationEndTime, frq=100):
        start_time=datetime.now()
        count=0
        holdTime=simulationEndTime/frq
        while 1:
            count+=1
            start_Inc_Time=datetime.now()
            yield hold, self, holdTime
            interval=datetime.now()-start_Inc_Time
            remaining=interval*(frq-count)
            remaining_minutes=int(remaining.seconds/60.0)
            runTime=datetime.now()-start_time
            runTime_minutes=int(runTime.seconds/60.0)
            print 'SimTime: %8.1f hours (Running: %d mins, Remaining: %d mins)' %(now(),
                runTime_minutes, remaining_minutes)

```

```

# Main Program Loop
def run_Prog():

    initialize()
    importData()

    global printToggle, operatorToggle,breakToggle
    global reworkToggle,scrapToggle,downtimeToggle
    global plotTSQ

    if gui.params.print_detail: printToggle=True
    else: printToggle=False

    if gui.params.Downtime: downtimeToggle=True
    else: downtimeToggle=False

    if gui.params.Operator: operatorToggle=True
    else: operatorToggle=False

    if gui.params.Rework: reworkToggle=True
    else: reworkToggle=False

    if gui.params.Scrap: scrapToggle=True
    else: scrapToggle=False

    if gui.params.Operator_Break: breakToggle=True
    else: breakToggle=False

    if gui.params.Plot_TSQ: plotTSQ=True
    else: plotTSQ=False

    global endtime,ct,exitedLots,exitMon,ctMon
    endtime=float(gui.params.Simulation_End_Time)
    ct=[];exitedLots=[]
    exitMon=Monitor()
    ctMon=Monitor()

    global fullLotScrap
    fullLotScrap=[]

    'tools'
    global toolsetQueue,toolList,incompleteBatch,tsqLevels,numToolGroups
    tools=[];toolsetQueue=[]; incompleteBatch=[];toolList=[];tsqLevels=[]

    numToolGroups=len(tsddata)

    for row in tsdata:
        desc=[];nr=[];ttr=[];oprID=[]
        for j in [6,11,16,21,26]: #the start of all dt columns
            if row[j+1]<>"":
                if int(row[j+1])==1: # run-based only
                    desc.append(row[j])
                    if stringToValChecker(row[j+2])>0.0: nr.append((float(row[j+2])/60))
                    else: nr.append(0.0)
                    if stringToValChecker(row[j+3])>0.0: ttr.append((float(row[j+3])/60))
                    else: ttr.append(0.0)
                    oprID.append(row[j+4])

        toolsetQueue.append(Store(name=row[0],
                                unitName='toolset',
                                capacity='unbounded',
                                initialBuffered=None,
                                monitored=True,
                                monitorType=Monitor))

        tsqLevels.append([row[0],0])

        for i in range(int(row[2])):
            t=toolSource()

            if len(nr)>0.0:
                tools.append(activate(t,t.generate(toolsetID=row[0],
                                                    toolID=i+1,
                                                    toolDescription=row[1],
                                                    dtDescription=desc,
                                                    numRunsBeforeFail=nr,
                                                    mtr=ttr,
                                                    dtoperatorsetID=oprID),
                                    at=0.0))
            else:
                tools.append(activate(t,t.generate(toolsetID=row[0],
                                                    toolID=i+1,

```

```

        toolDescription=row[1],
        dtDescription=0,
        numRunsBeforeFail=0,
        mttr=0,
        dtoperatorsetID=0),
        at=0.0))

'downtime'
global downtimeQueue, dtList
downtimes=[]; downtimeQueue=[]; dtList=[]

if downtimeToggle:
    for row in tsdata:
        desc=[]; tbf=[]; ttr=[]; oprtID=[]
        for j in [6,11,16,21,26]: #the start of all dt columns
            if row[j+1]<>"":
                if int(row[j+1])==0: # time-based only
                    desc.append(row[j])
                    if stringToValChecker(row[j+2])>0.0: tbf.append((float(row[j+2])/60))
                    else: tbf.append(0.0)
                    if stringToValChecker(row[j+3])>0.0: ttr.append((float(row[j+3])/60))
                    else: ttr.append(0.0)
                    oprtID.append(row[j+4])

        d=downtimeSource()
        for i in range(int(row[2])):
            downtimeQueue.append(SimEvent(row[0]+"-"+str(i+1)))
            d=downtimeSource()
            downtimes.append(activate(d,d.generate(toolsetID=row[0],
                toolID=i+1,
                dtDescription=desc,
                dtMTBF=tbf,
                dtMTR=ttr,
                dtOperatorID=oprtID),
                at=0.0))

'operators'
if operatorToggle:
    global operatorsetQueue, operatorList, numOperatorGroups
    ops=[]; operatorsetQueue=[]; operatorList=[]
    numOperatorGroups=len(osdata)

    for row in osdata:
        operatorsetQueue.append(Store(name=row[0],
            unitName='opertorset',
            capacity='unbounded',
            initialBuffered=None,
            monitored=True,
            monitorType=Monitor))
        for i in range(int(row[2])):
            op=operatorSource()
            ops.append(activate(op,op.generate(n=int(i+1),
                op_setID=row[0],
                op_Desc=row[1]),
                at=0.0))

'breaks'
if breakToggle:
    global breakList, breakQueue
    breakList=[]; breaks=[]; breakQueue=[]
    for row in osdata:
        breakDescription=[]; timeBetweenBreak=[]; breakLength=[]
        for j in [3,6,9]: # the start of all break columns
            breakDescription.append(row[j])
            if row[j+1]<>"": timeBetweenBreak.append(float(row[j+1])/60)
            else: timeBetweenBreak.append(0.0)
            if row[j+2]<>"": breakLength.append(float(row[j+2])/60)
            else: breakLength.append(0.0)

        for i in range(int(row[2])):
            br=breakSource()
            breakQueue.append(SimEvent(row[0]+"-"+str(i+1)))
            breaks.append(activate(br,br.generate(n=int(i+1),
                op_setID=row[0],
                break_Desc=breakDescription,
                betweenBreaks=timeBetweenBreak,
                bl=breakLength),
                at=0.0))

'lots'
global lotList
recipes=[]; lotList=[]
for proc in vrdata:

```

```

s=lotSource()
nLotsDay=float(float(proc[3])/int(proc[4]))
iaTime=24/nLotsDay
recipes.append(activate(s,s.generate(process_Flow_ID=proc[0],
                                   product_ID=proc[1],
                                   product_Name=proc[2],
                                   lot_Size=int(proc[4]),
                                   inter_Arrival_Time=iaTime), # daily release
                                   at=0.0))
#break # nbedit for RPT to release the first product only

'begin model'
tm=timer()
activate(tm,tm.start(endtime))
simulate(until=endtime)

'output'
if len(exitedLots)>0:
    tempSum=0
    for lt in exitedLots:
        tempSum=tempSum+lt.ct

    print 'number of lots entered is ',len(lotList)
    print 'number of lots exited is ',len(exitedLots)
    print 'average cycle time is', tempSum / len(exitedLots)
else:
    print 'no lots exited the system'

print 'number of lots scrapped is ',len(fullLotScrap)

if operatorToggle:
    print 'Operator Use'
    for oprID in range(numOperatorGroups):
        numOpr=0
        sumU=0
        for opr in operatorList:
            if (opr.operatorsetID==str(oprID+1)):
                numOpr += 1
                sumU += opr.runTime

        print 'OP%d inUse=%4.2f' %(oprID+1,(sumU/numOpr)/endtime)

toolUse=[]
for tsID in range(numToolGroups):
    toolUse.append(0)
    numT=0
    sumU=0
    for t in toolList:
        if (t.toolsetID==str(tsID+1)):
            numT += 1
            sumU += t.runTime
    toolUse[tsID]=((sumU/numT)/endtime)

if downtimeToggle:
    print 'Tools'
    dtUse=[]
    for tsID in range(numToolGroups):
        dtUse.append(0)
        numDT=0
        sumU=0
        for dt in dtList:
            if (dt.toolsetID==str(tsID+1)):
                numDT += 1
                sumU += dt.runTime

        if numDT>0:
            dtUse[tsID]=((sumU/numDT)/endtime)

if downtimeToggle:
    for i in range(len(dtUse)):
        print 'TS%d inProc=%6.4f Down=%6.4f u/a=%6.4f' %(i+1,toolUse[i],dtUse[i], toolUse[i]/(1.0-dtUse[i]))
    else:
        for i in range(len(toolUse)):
            print 'TS%d inUse=%6.4f' %(i+1,toolUse[i])

"Plot CT"
plt=SimPlot()
plt.plotLine(ctMon,color='blue')
plt.plotLine(exitMon,color='red')

```

```

plt.mainloop()

# Lot Components
class lotSource(Process):
    "source generates lots"

    def generate(self,
                 process_Flow_ID,
                 product_ID,
                 product_Name,
                 lot_Size,
                 inter_Arrival_Time):

        count=0
        while now() <endtime:
            count+=1
            l=lot(n=count,
                 processFlowID=process_Flow_ID,
                 productID=product_ID,
                 productName=product_Name,
                 lotSize=lot_Size)

            activate(l,l.doSteps()) # main loop called here
            yield hold, self, expovariate(1/inter_Arrival_Time)
            #yield hold, self, endtime #for rpt (Nbedit)

class lot(Process):
    "lots"

    def __init__(self, n,
                 processFlowID,
                 productID,
                 productName,
                 lotSize):

        Process.__init__(self, name=productID + "-Lot" +str(n))
        self.processFlowID = processFlowID
        self.productID = productID
        self.productName = productName
        self.lotSize = lotSize
        self.stepList=[]
        self.startTime=now()
        self.operationDescription=""
        self.toolsetID=""
        self.operatorsetID=""
        self.loadTime=0.0
        self.unloadTime=0.0
        self.timePerWaferInProcess=0.0
        self.waferTravelTime=0.0
        self.timePerLot=0.0
        self.timePerBatch=0.0
        self.minBatchsize=0
        self.maxBatchsize=0
        self.batchID=""
        self.timePerSpecSetup=0.0
        self.timePerGroupSetup=0.0
        self.setupGroupID=""
        self.lotScrapProb=0.0
        self.waferScrapProb=0.0
        self.lotReworkProb=0.0
        self.waferReworkProb=0.0
        self.ReworkSequenceID=0.0
        self.reworkReturnStepID=""
        self.travelTime=0.0
        self.travelTimeOperatorID=""
        self.toolDescription=""
        self.operatorLoadFraction=0.0
        self.operatorProcessFraction=0.0
        self.operatorUnloadFraction=0.0
        self.currentStep=""
        self.reqBatch=0
        self.batchSize=0
        self.originalLotSize=0
        self.batchList=[]
        self.batchName=""
        self.batchFlag=False
        self.username=""
        self.reworkFlag=False
        self.currentRWStep=""
        self.reworkStepList=""
        self.secondaryLot=False
        self.type='lot'
        self.startStepTime=0.0

```

```

self.stepCT=[]
self.ct=0.0
lotList.append(self)

def doSteps(self):
    """carries out all the steps in a lots process flow"""

    returnFlag=True #controls lots that need to return to earlier step

    for row in prdata:
        if self.processFlowID==row[0]:
            self.stepList.append(row[1])

    while returnFlag:
        returnFlag=False
        for theStep in self.stepList:
            self.currentStep=theStep
            self.startStepTime=now()
            self.getDetail()
            self.getToolDetail()
            self.secondaryLot=False
            self.batchFlag=False

            if printToggle: print '%8.4f :'%(now()), self.name, 'has begun step', self.currentStep

            # Update toolset queue for plotting
            if plotTSQ:
                for tsp in tsqLevels:
                    if tsp[0]==self.toolsetID:
                        tsp[1]=int(tsp[1])+1
                        break
                updateTSQPlot()

            # Batching
            if self.maxBatchsize>0 and self.minBatchsize>0:
                self.reqBatch=self.minBatchsize # or minBatchSize
                lotsToBatch=[]
                batchName=[]
                self.batchList=[]
                batchCount=0

                if self.batchID <>"":
                    'batch by batchID'
                    for bt in incompleteBatch:
                        if (bt.batchID==self.batchID and
                            bt.toolsetID==self.toolsetID):
                            batchCount+=1
                            lotsToBatch.append(bt)
                            batchName.append(bt.name)
                else:
                    'batch by process flow and step ID'
                    for bt in incompleteBatch:
                        if (bt.reworkFlag==False and
                            bt.processFlowID==self.processFlowID and
                            bt.currentStep==self.currentStep and
                            bt.toolsetID==self.toolsetID):
                            batchCount+=1
                            lotsToBatch.append(bt)
                            batchName.append(bt.name)

            if self.reqBatch==batchCount+1:
                'batch is complete'
                batchName.insert(0,self.name)
                self.batchName=batchName
                self.batchFlag=True
                self.batchSize=batchCount+1
                self.originalLotSize=self.lotSize

            for lt in lotsToBatch:
                self.lotSize=self.lotSize+lt.lotSize
                self.batchList.append(lt)
                self.interrupt(lt)
                'remove from incompleteBatch list'
                foundLoc=False
                for r, bt in enumerate(incompleteBatch):
                    if lt.name==bt.name:
                        loc=r
                        foundLoc=True
                        break

            if foundLoc:
                incompleteBatch.pop(loc)

```

```

    else: print "warning did not find lot",lt.name

    if printToggle: print '%8.4f :'%(now()), self.batchName, \
        'formed a batch for toolset',self.toolsetID

else:
    'batch is incomplete join the batch queue'
    if printToggle: print '%8.4f :'%(now()),self.name, \
        'is waiting to batch for toolset', self.toolsetID
    self.batchFlag=True
    incompleteBatch.append(self)
    'hold the lot until the batch is completed'
    yield hold, self, endtime
    if self.interrupted(): # as in batch formed
        "hold until primary lot in batch is finished its operation"
        self.interruptReset()
        yield hold, self, endtime
    if self.interrupted(): # primary lot in batch is finished
        self.interruptReset()
        self.batchFlag=False
        self.secondaryLot=True # so it skips to unbatching

if self.secondaryLot==False: # i.e. a normal lot or a primary batch lot

    if self.batchFlag: self.username=self.batchName
    else: self.username=self.name

    # Queueing
    for ts in toolsetQueue:
        if ts.name==self.toolsetID:
            if printToggle: print '%8.4f :'%(now()), self.username, \
                'is waiting for an available tool on toolset', ts.name
            yield get, self, ts, 1, 1
            toolUsed=self.got[0]
            toolCapturedAt=now()
            if printToggle: print '%8.4f :'%(now()), self.username, \
                'has captured tool', toolUsed.name
            break

    # Update toolset queue for plotting
    if plotTSQ:
        if self.batchFlag:
            tsp[1]=tsp[1]-(self.reqBatch)
        else:
            tsp[1]=tsp[1]-1

        updateTSQPlot()

    # Setup
    if self.timePerSpecSetup>0.0:
        if (self.currentStep<>toolUsed.lastStep or
            self.processFlowID<>toolUsed.lastProcessFlowID):
            if printToggle: print '%8.4f :'%(now()), toolUsed.name, \
                'is changing setup for different process flow step'
            yield hold, self, self.timePerSpecSetup
            if printToggle: print '%8.4f :'%(now()), toolUsed.name, \
                'has completed setup for',self.currentStep,'on',self.username

    if self.timePerGroupSetup>0.0 and toolUsed.lastSetupGroupID<>"":
        if self.setupGroupID<>toolUsed.lastSetupGroupID:
            if printToggle: print '%8.4f :'%(now()), toolUsed.name, \
                'is changing setup for setup group ID',self.setupGroupID
            yield hold, self, self.timePerGroupSetup
            if printToggle: print '%8.4f :'%(now()), toolUsed.name, \
                'has completed setup for',self.currentStep,'on',self.username

    toolUsed.lastProcessFlowID=self.processFlowID
    toolUsed.lastStep=self.currentStep
    toolUsed.lastSetupGroupID=self.setupGroupID

    # Check Operator
    operatorReq=False
    if operatorToggle and self.operatorsetID <> "":
        for os in operatorsetQueue:
            if os.name==self.operatorsetID:
                operatorReq=True
                break

    if self.loadTime>0.0:
        if operatorToggle and operatorReq and self.operatorLoadFraction>0.0:
            'get an operator for a loading'
            if printToggle: print '%8.4f :'%(now()), self.username, 'is waiting for operator
                set',\

```



```

        self.operatorsetID, 'to load on tool', toolUsed.name
    yield get, self, os, 1, 1
    operatorGotAt=now()
    operatorUsed=self.got[0]
    if printToggle: print '%8.4f :%(now()), 'Operator', operatorUsed.name, 'is
        loading', \
        self.username, 'on tool', toolUsed.name

    'hold operator for a set time'
    yield hold, self, self.operatorLoadFraction*float(self.loadTime)
    if printToggle: print '%8.4f :%(now()), 'Operator', operatorUsed.name, \
        'is finished loading', self.username, 'on', toolUsed.name

    'release the operator'
    if printToggle: print '%8.4f :%(now()), 'Operator', operatorUsed.name, 'is
        finished loading', \
        self.username, 'on tool', toolUsed.name
    operatorUsed.runTime=operatorUsed.runTime + (now()-operatorGotAt)
    operatorUsed.doneSignal.signal('lot')

    'hold the lot for the remaining loading time'
    yield hold, self, (1.0-self.operatorLoadFraction)*self.loadTime #remainder
    if printToggle: print '%8.4f :%(now()), self.username, \
        'is finished loading on', toolUsed.name

else:
    "we dont need an loading operator"
    if printToggle: print '%8.4f :%(now()), self.username, 'is loading on tool',
        toolUsed.name
    yield hold, self, self.loadTime

# Processing
pt = (self.timePerBatch +
    self.timePerLot +
    self.timePerWaferInProcess * self.lotSize)
if pt > 0.0:
    if operatorToggle and operatorReq and self.operatorProcessFraction>0.0:
        'get an operator for processing'
        if printToggle: print '%8.4f :%(now()), self.username, 'is waiting for operator
            set', \
            self.operatorsetID, 'for processing on tool', toolUsed.name
        yield get, self, os, 1, 1
        operatorGotAt=now()
        operatorUsed=self.got[0]
        if printToggle: print '%8.4f :%(now()), 'Operator', operatorUsed.name, 'is
            processing', \
            self.username, 'on tool', toolUsed.name

        'hold operator for a set time'
        yield hold, self, self.operatorProcessFraction*pt
        if printToggle: print '%8.4f :%(now()), 'Operator', operatorUsed.name, \
            'is finished processing', self.username, 'on', toolUsed.name

        'release the operator'
        if printToggle: print '%8.4f :%(now()), 'Operator', operatorUsed.name, 'is
            finished processing', \
            self.username, 'on tool', toolUsed.name

        operatorUsed.runTime=operatorUsed.runTime + (now()-operatorGotAt)
        operatorUsed.doneSignal.signal('lot')

        'hold the lot for the remaining loading time'
        yield hold, self, (1.0-self.operatorProcessFraction)*pt #remainder
        if printToggle: print '%8.4f :%(now()), self.username, \
            'is finished processing on', toolUsed.name

else:
    "we dont need an operator to process"
    if printToggle: print '%8.4f :%(now()), self.username, 'is processing on tool',
        toolUsed.name
    yield hold, self, pt
    if printToggle: print '%8.4f :%(now()), self.username, \
        'is finished processing on tool', toolUsed.name

#Unloading
if self.unloadTime>0.0:
    if operatorToggle and operatorReq and self.operatorUnloadFraction>0.0:
        'get an operator for a loading'
        if printToggle: print '%8.4f :%(now()), self.username, 'is waiting for operator
            set', \
            self.operatorsetID, 'to unload from tool', toolUsed.name
        yield get, self, os, 1, 1

```

```

operatorUsed=self.got[0]
operatorGotAt=now()
if printToggle: print '%8.4f :%(now()), 'Operator', operatorUsed.name, 'is
unloading', \
self.username, 'from tool', toolUsed.name

'hold operator for a set time'
yield hold,self,self.operatorUnloadFraction*float(self.unloadTime)
if printToggle: print '%8.4f :%(now()), 'Operator', operatorUsed.name, \
'is finished unloading', self.username,'from', toolUsed.name

'release the operator'
if printToggle: print '%8.4f :%(now()), 'Operator', operatorUsed.name, 'is
finished unloading', \
self.username, 'from tool', toolUsed.name
operatorUsed.runTime=operatorUsed.runTime +(now()-operatorGotAt)
operatorUsed.doneSignal.signal('lot')

'hold the lot for the remaining unloading time'
yield hold,self,(1.0-self.operatorUnloadFraction)*self.unloadTime #remainder
if printToggle: print '%8.4f :%(now()), self.username, \
'is finished unloading from', toolUsed.name
else:
"we dont need an unloading operator"
if printToggle: print '%8.4f :%(now()), self.username, 'is unloading from tool',
toolUsed.name
yield hold, self, self.unloadTime

# Tool Release
if printToggle: print '%8.4f :%(now()), self.username, 'is finished on tool',
toolUsed.name
toolUsed.runTime=toolUsed.runTime+(now()-toolCapturedAt)
toolUsed.doneSignal.signal('lot')

# Wafer Travel
if self.waferTravelTime>0.0:
if printToggle: print '%8.4f :%(now()), self.username, \
'has begun wafer travel for step', self.currentStep
yield hold, self,self.waferTravelTime

# Unbatching
if self.batchFlag==True:
'send signal to release the secondary lots in the batch'
for ml in self.batchList:
self.interrupt(ml)

self.batchType=""
self.batchFlag=False
self.batchName=""
self.batchList=[]
self.lotSize=self.originalLotSize
self.batchSize=0
if printToggle: print '%8.4f :%(now()), self.username, 'is unbatching'

'secondary lots in a batch jump to here'
# Lot Transport
if self.travelTime>0.0:
if operatorToggle and self.travelTimeOperatorID <> "":
'gets an operator for lot transport'
for os in operatorsetQueue:
if os.name==self.travelTimeOperatorID:
break

if printToggle: print '%8.4f :%(now()), self.name, 'is waiting for operator set',
self.operatorsetID, 'to transport from',self.currentStep
yield get, self,os,1,1
operatorUsed=self.got[0]
if printToggle: print '%8.4f :%(now()), 'Operator', operatorUsed.name, 'is
transporting', \
self.name, 'from step', self.currentStep

'hold operator for a set time'
yield hold, self, self.travelTime
if printToggle: print '%8.4f :%(now()), 'Operator', operatorUsed.name, \
'is finished transporting',self.name,'from step', self.currentStep

'release the operator'
operatorUsed.doneSignal.signal('lot')
else:
"we dont need an operator for lot transport"
if printToggle: print '%8.4f :%(now()), self.name, \
'is transporting from step', self.currentStep

```

```

        yield hold, self, self.travelTime
        if printToggle: print '%8.4f :'%(now()), self.name, \
            'is finished transporting from step', self.currentStep

# Scrap Test
if scrapToggle and self.lotScrapProb>0.0:
    if random.random()<self.lotScrapProb:
        if printToggle: print '%8.4f :'%(now()), self.name, \
            'failed lot scrap test and exited'
        fullLotScrap.append(self)
        yield passivate,self
        break

if scrapToggle and self.waferScrapProb>0.0:
    numWfrScrap=0
    for wfr in range(self.lotSize): #@UnusedVariable
        if random.random()< self.waferScrapProb:
            numWfrScrap+=1
    if numWfrScrap>0:
        if self.lotSize-numWfrScrap > 0:
            self.lotSize=self.lotSize-numWfrScrap
            if printToggle: print '%8.4f :'%(now()),numWfrScrap,\
                'wafers scrapped from',self.name
        else: #all wafers scrapped
            if printToggle: print '%8.4f :'%(now()),\
                'all remaining wafers were scrapped from',self.name
            fullLotScrap.append(self)
        yield passivate,self

# Rework Test
if reworkToggle and self.lotReworkProb>0.0:
    if random.random()<self.lotReworkProb:
        'not implememnted yet'

if reworkToggle and self.waferReworkProb>0.0:
    numWfrRwk=0
    for wfr in range(self.lotSize): #@UnusedVariable
        if random.random()< self.waferReworkProb:
            numWfrRwk+=1
    if numWfrRwk>0:
        """ not implememnted yet"""

# finished step
self.stepCT.append(now()-self.startStepTime)
if printToggle: print '%8.4f :'%(now()), self.name, \
    'is finished step', self.currentStep

#print '%8.4f :'%(now()-self.startStepTime),self.currentStep #for rpt checking

#Exit
useS=0
if returnFlag==True:
    for s in range(len(self.stepList)):
        if self.stepList[s]==self.reworkReturnStepID:
            useS=s
            break
    'remove the completed steps'
    useS=useS-1
    for inc in range(useS):
        self.stepList.popleft()

self.ct=(now()-self.startTime)
exitedLots.append(self)
exitMon.observe(self.ct)
ctMon.observe(exitMon.mean())

if printToggle: print '%8.4f :'%(now()), self.name, \
    'has exited the system after',now()-self.startTime,'hours'

def doRWSteps(self):
    """carries out all the rework steps in a lots process flow"""

    self.reworkFlag=True
    for row in rwdata:
        if self.reworkSequenceID==row[0]:
            self.reworkStepList.append(row[1])

    for theStep in self.reworkStepList:
        self.currentRWStep=theStep
        self.getreworkDetail()
        self.getToolDetail()

```

```

    if printToggle: print '%8.4f :'%(now()), self.name, 'has begun rework step', self.
        currentRWStep
    break

def getreworkDetail(self):
    foundFlag=False
    for row in rwdata:
        if self.ReworkSequenceID==row[0]:
            self.operationDescription=row[2]
            self.toolsetID=row[3]
            self.operatorsetID=row[4]
            self.reworkFlag=True
            if stringToValChecker(row[5]): self.loadTime=float(row[5])/60
            if stringToValChecker(row[6]): self.unloadTime=float(row[6])/60
            if stringToValChecker(row[7]): self.timePerWaferInProcess=float(row[7])/60
            if stringToValChecker(row[8]): self.waferTravelTime=float(row[8])/60
            if stringToValChecker(row[9])>0: self.timePerLot=float(row[9])/60
            if stringToValChecker(row[10])>0: self.timePerBatch=float(row[10])/60
            if stringToValChecker(row[11])>0: self.minBatchsize=int(ceil(float(row[11])/self.
                lotSize))
            if stringToValChecker(row[12])>0: self.maxBatchsize=int(ceil(float(row[12])/self.
                lotSize))
            self.batchID=row[13]
            if stringToValChecker(row[14]): self.timePerSpecSetup=float(row[14])/60
            if stringToValChecker(row[15]): self.timePerGroupSetup=float(row[15])/60
            self.setupGroupID=row[16]
            self.LotScrapProb=float(row[17])
            self.waferScrapProb=float(row[18])
            self.lotReworkProb=float(row[19])
            if stringToValChecker(row[23]): self.travelTime=float(row[23])/60
            self.travelTimeOperatorID=row[24]
            foundFlag=True
            break

    if foundFlag==False:
        if printToggle: print ""Warning could not find details in the
            rework route data for "" + self.name

def getDetail(self):
    for row in prdata:
        if self.processFlowID==row[0] and self.currentStep==row[1]:
            self.operationDescription=row[2]
            self.toolsetID=row[3]
            self.operatorsetID=row[4]
            if stringToValChecker(row[5]): self.loadTime=float(row[5])/60
            else: self.loadTime=0.0
            if stringToValChecker(row[6]): self.unloadTime=float(row[6])/60
            else: self.unloadTime=0.0
            if stringToValChecker(row[7]): self.timePerWaferInProcess=float(row[7])/60
            else: self.timePerWaferInProcess=0.0
            if stringToValChecker(row[8]): self.waferTravelTime=float(row[8])/60
            else: self.waferTravelTime=0.0
            if stringToValChecker(row[9]): self.timePerLot=float(row[9])/60
            else: self.timePerLot=0.0
            if stringToValChecker(row[10]): self.timePerBatch=float(row[10])/60
            else: self.timePerBatch=0.0
            if stringToValChecker(row[11]): self.minBatchsize=int(ceil(float(row[11])/self.lotSize
                ))
            else: self.minBatchsize=0
            if stringToValChecker(row[12]): self.maxBatchsize=int(ceil(float(row[12])/self.lotSize
                ))
            else: self.maxBatchsize=0
            self.batchID=row[13]
            if stringToValChecker(row[14]): self.timePerSpecSetup=float(row[14])/60
            else: self.timePerSpecSetup=0.0
            if stringToValChecker(row[15]): self.timePerGroupSetup=float(row[15])/60
            else: self.timePerGroupSetup=0.0
            self.setupGroupID=row[16]
            if stringToValChecker(row[17]): self.lotScrapProb=float(row[17])
            else: self.lotScrapProb=0.0
            if stringToValChecker(row[18]): self.waferScrapProb=float(row[18])
            else: self.waferScrapProb=0.0
            if stringToValChecker(row[19]): self.lotReworkProb=float(row[19])
            else: self.lotReworkProb=0.0
            if stringToValChecker(row[20]): self.waferReworkProb=float(row[20])
            else: self.waferReworkProb=0.0
            self.ReworkSequenceID=row[21]
            self.reworkReturnStepID=row[22]
            if stringToValChecker(row[23]): self.travelTime=float(row[23])/60
            else: self.travelTime=0.0
            self.travelTimeOperatorID=row[24]
            self.batchType=""
            foundFlag=True

```

```

        break

    if foundFlag==False:
        if printToggle: print ""Warning could not find details in the
            process route data for "" + self.name

def getToolDetail(self):
    foundFlag=False
    for row in tsdata:
        if self.toolsetID==row[0]:
            self.toolDescription=row[1]
            if stringToValChecker(row[3]): self.operatorLoadFraction=float(row[3])
            else: self.operatorLoadFraction=0.0
            if stringToValChecker(row[4]): self.operatorProcessFraction=float(row[4])
            else: self.operatorProcessFraction=0.0
            if stringToValChecker(row[5]): self.operatorUnloadFraction=float(row[5])
            else: self.operatorUnloadFraction=0.0
            foundFlag=True
            break

    if foundFlag==False:
        if printToggle: print "Warning could not find details in the tool set data for ",\
            self.name," that requires tool set" , self.toolsetID

# Tool Components
class toolSource(Process):

    def generate(self,
                toolsetID,
                toolID,
                toolDescription,
                dtDescription,
                numRunsBeforeFail,
                dtoperatorsetID,
                mttr):

        t=tool(toolset_ID=toolsetID,
              tool_ID=toolID,
              tool_Description=toolDescription,
              dt_Description=dtDescription,
              num_Runs_Before_Fail=numRunsBeforeFail,
              dt_Operator_ID=dtoperatorsetID,
              time_To_Repair=mttr)

        activate(t,t.run())
        yield hold, self,endtime

class tool(Process):
    def __init__(self,
                toolset_ID,
                tool_ID,
                tool_Description,
                dt_Description,
                num_Runs_Before_Fail,
                dt_Operator_ID,
                time_To_Repair):

        Process.__init__(self,name=toolset_ID + "-" + str(tool_ID))
        toolList.append(self)
        self.toolsetID=toolset_ID
        self.toolID=tool_ID
        self.toolDescription=tool_Description
        self.lastProcessFlowID=""
        self.lastStep=""
        self.runTime=0
        self.occupied=False
        self.down=False
        self.lastSetupGroupID=""
        self.type='tool'
        self.doneSignal=SimEvent()

    if num_Runs_Before_Fail>0.0:
        self.rbdFlag=True
        self.rbdtDescription=dt_Description
        self.mrbf=num_Runs_Before_Fail
        self.ttr=time_To_Repair
        self.dtOperatorsetID=dt_Operator_ID

    temp=[]
    for i in range(len(self.rbdtDescription)): #@UnusedVariable
        temp.append(0)
    self.runCount=temp

```

```

else:
    self.rbdtdescription=""
    self.mrbf=0
    self.ttr=0
    self.runCount=0
    self.dtOperatorsetID=0
    self.rbdtdFlag=False

def run(self):

    for ts in toolsetQueue:
        if ts.name == self.toolsetID:
            break

    while True:
        yield put, self, ts, [self] #put it in the toolsetQueue store
        yield waitevent, self, self.doneSignal
        heldBy=self.doneSignal.signalparam

        if heldBy=='lot':
            self.runCount+=1
            'check for run-based downtime'
            if downtimeToggle and self.rbdtdFlag:
                for i in range(len(self.runCount)):
                    if self.runCount[i]>=self.mrbf[i]:
                        if printToggle: print '%8.4f :'%(now()), 'Tool', \
                            self.name, 'has gone down for run based downtime', \
                            self.rbdtdescription[i]
                        yield hold, self, self.ttr[i]
                        self.runCount[i]=0
                        if printToggle: print '%8.4f :'%(now()), 'Tool', \
                            self.name, 'has recovered from run based downtime', \
                            self.rbdtdescription[i]

# Downtime Components
class downtimeSource(Process):
    "source generates downtime units"

    def generate(self,
                 toolsetID,
                 toolID,
                 dtDescription,
                 dtMTBF,
                 dtMTTR,
                 dtOperatorID):

        for i in range(len(dtDescription)):
            if dtMTBF[i]>0.0 and dtMTTR[i]>0.0:
                dt=downtime(tool_ID=toolID,
                           toolset_ID=toolsetID,
                           dt_Description=dtDescription[i],
                           dt_MTBF=dtMTBF[i],
                           dt_MTTR=dtMTTR[i],
                           dt_OperatorID=dtOperatorID[i])
                activate(dt, dt.run())

        yield hold, self, endtime

class downtime(Process):
    "downtime item initialisation"

    def __init__(self,
                 toolset_ID,
                 tool_ID,
                 dt_Description,
                 dt_MTBF,
                 dt_MTTR,
                 dt_OperatorID):

        Process.__init__(self, name=toolset_ID + "-" + str(tool_ID))
        dtList.append(self)
        self.toolsetID=toolset_ID
        self.toolID=tool_ID
        self.Description=dt_Description
        self.mttr=dt_MTTR
        self.mtbf=dt_MTBF
        self.runTime=0
        self.dtoperatorsetID=dt_OperatorID
        self.type='downtime'

    def run(self):
        "causes downtime on tools"

```

```

def getItem(buff):
    "gets the tool from the toolsetQueue store"
    result=[]
    for item in buff:
        if item.name==self.name:
            result.append(item)
            break
    return result

for ts in toolsetQueue:
    if ts.name==self.toolsetID:
        break

while True:
    mtbfSample=expovariate(1/self.mtbf)
    yield hold, self, mtbfSample # downtime item waits for mtbf

    'get the tool'
    yield get, self, ts, getItem, 3 #priority is 3
    toolGotAt=now()
    toolOccupied=self.got[0]
    if printToggle: print '%8.4f :'%(now()), 'Tool', \
        toolOccupied.name, 'has gone down for', self.Description

    'locate the downtime operator'
    if operatorToggle and self.doperatorsetID<>"":
        for os in operatorsetQueue:
            if os.name==self.doperatorsetID:
                break

        if printToggle: print '%8.4f :'%(now()), 'Waiting for operator set', \
            self.doperatorsetID, 'to repair', toolOccupied.name
        yield get, self, os, 1, 2
        operatorGotAt=now()
        operatorUsed=self.got[0]
        if printToggle: print '%8.4f :'%(now()), 'Operator', operatorUsed.name, \
            'is repairing tool', toolOccupied.name

    mtrrSample=expovariate(1/self.mtrr)
    yield hold, self, mtrrSample
    self.runTime=self.runTime+(now()-toolGotAt)
    toolOccupied.doneSignal.signal('downtime')

    'release the repair operator'
    if operatorToggle and self.doperatorsetID<>"":
        operatorUsed.doneSignal.signal('downtime')
        operatorUsed.runTime=operatorUsed.runTime+(now()-operatorGotAt)
        if printToggle: print '%8.4f :'%(now()), 'Operator', operatorUsed.name, \
            'is finished repairing tool', toolOccupied.name

    if printToggle: print '%8.4f :'%(now()), 'Tool', \
        toolOccupied.name, 'is back online'

# Operator Components
class operatorSource(Process):
    "source generates operators"

    def generate(self, n,
                op_setID,
                op_Desc):

        op=operator(num=n,
                    opSetID=op_setID,
                    opDesc=op_Desc)

        activate(op, op.run())

        yield hold, self, endtime

class operator(Process):
    "operator item initialisation"

    def __init__(self,
                num,
                opSetID,
                opDesc):

        Process.__init__(self, name=opSetID + "-" + str(num))
        operatorList.append(self)
        self.operatorsetID=opSetID
        self.operatorDescription=opDesc
        self.occupied=False

```

```

self.runTime=0
self.type='operator'
self.doneSignal=SimEvent()

def run(self):
    "stasis loop for operators (slaves)"

    for os in operatorsetQueue:
        if os.name == self.operatorsetID:
            break

    while True:
        yield put, self, os, [self] #put it in the operatorsetQueue store
        yield waitevent, self, self.doneSignal

# Break Components
class breakSource(Process):
    "source generates breaks"

    def generate(self, n,
                 op_setID,
                 break_Desc,
                 betweenBreaks,
                 bl):

        for i in range(3):
            if betweenBreaks[i]>0.0 and bl[i]>0.0:
                br=breakItem(num=n,
                             opSetID=op_setID,
                             breakDesc=break_Desc[i],
                             between_breaks=float(betweenBreaks[i]),
                             break_Length=bl[i])

                activate(br, br.run())

        yield hold, self, endtime

class breakItem(Process):
    "breakitem initialisation"

    def __init__(self,
                 num,
                 opSetID,
                 breakDesc,
                 between_breaks,
                 break_Length):

        Process.__init__(self, name=opSetID + "-" + str(num))
        self.operatorsetID=opSetID
        self.breakDescription=breakDesc
        self.tbb=between_breaks
        self.bl=break_Length
        self.type='break'
        breakList.append(self)

    def run(self):
        "stasis loop for break items (masters)"

        def getItem(buff):
            "gets the operator from the operatorsetQueue store"
            result=[]
            for item in buff:
                if item.name==self.name:
                    result.append(item)
                    break
            return result

        for os in operatorsetQueue:
            if os.name==self.operatorSetID:
                break

        while True:
            tbbSample=self.tbb
            yield hold, self, tbbSample
            yield get, self, getItem, 3
            operatorCaptured=self.got[0]
            if printToggle: print '%8.4f :'%(now()), 'Operator', \
                operatorCaptured.name, 'is breaking for', self.breakDescription

            'send operator for break'
            blSample=self.bl
            yield hold, self, blSample

```



```
operatorCaptured.doneSignal.signal('break')
if printToggle: print '%8.4f :'%(now()), 'Operator', \
    self.name,'has finished', self.breakDescription, 'break'

# GUI
class SDSGUI(SimGUI):
    def __init__(self,win,**par):
        SimGUI.__init__(self,win,**par)
        self.run.add_command(label="Run Model",
            command=run_Prog,underline=0)
        self.params=Parameters(Dataset=1,
            print_detail=0,
            Simulation_End_Time=50000,
            Downtime=1,
            Operator=1,
            Rework=0,
            Scrap=0,
            Plot_TSQ=1,
            Operator_Break=0)

# Open GUI
root=Tk()
gui=SDSGUI(root,title="Sematech Simulation Console",consoleHeight=20)
gui.writeConsole("Click Run -> Run Model.To change parameters click Edit -> Change Parameters"
)
gui.mainloop()
```

Johnson Distribution

There are a number of distributions appropriate for characterising the *time to perform a task* such as the lognormal, beta and logistic (Altiok, 1997; Law and Kelton, 1997; Taha, 2005). Storer et al. (1988) and Wheeler (1980) suggest that the Johnson family of distributions is one of the most generally applicable distribution types for this purpose. Johnson distributions are transformations of the normal distribution and can be used to describe most naturally occurring uni-modal sets of data (DeBroda et al., 1989). The Johnson family offers three curve fits; the Johnson unbounded S_U , Johnson bounded S_B and the Johnson lognormal S_L distribution (Slifker and Shapiro, 1980).

Selection of the most applicable of these distributions is based on the availability and credibility of the data sample. If the data sample is large and verified, then the unbounded distribution is used. The bounded distribution is generally better for smaller sample sizes. The Johnson lognormal distribution is used in very unique circumstances that exist in a narrow range between selection of the bounded and the unbounded, and

requires an interval of operation that is poorly-defined.

F.1 Algorithm

The selection procedure for this method is given by Slifker and Shapiro (1980). The algorithm delivers estimates for the four parameters required to describe the Johnson distribution; the starting location ϵ , the range λ , the skewness γ and a shape factor η .

The normal standard variable z is given by the transformation $z = \gamma + \eta k_i(x, \lambda, \epsilon)$ for a sample set of data points from a population $x_1, x_2, x_3, \dots, x_r$ where $r > 10$. The choice of z is motivated by the number of data points from the sample. For moderate sized datasets ($10 < r \leq 25$), z should be less than 1.0; hence the selection of $z = 0.5$ in the algorithm. Choosing a value greater than 1.0 would make it difficult to estimate the points in the higher percentile. However, for larger datasets ($r > 25$) a larger z value is allowed, hence $z = 1.2$ is used in those circumstances.

For $\zeta = -3z, -z, z, 3z$, the normal distribution from the percentages is given by P_ζ . The corresponding percentile $x^{(i)}$ to P_ζ is the i^{th} ordered observation where $i = rP_\zeta + 1/2$ which gives x_{-3z}, x_{-z}, x_z and x_{3z} .

$$m = x_{3z} - x_z \tag{F.1}$$

$$n = x_{-z} - x_{-3z} \tag{F.2}$$

$$p = x_z - x_{-z} \tag{F.3}$$

$$\frac{mn}{p^2} > 1 \Rightarrow S_U \quad (\text{F.4})$$

$$\frac{mn}{p^2} \approx 1 \Rightarrow S_L \quad (\text{F.5})$$

$$\frac{mn}{p^2} < 1 \Rightarrow S_B \quad (\text{F.6})$$

From Eqs.(F.1)-(F.3), the quantile sizes m , n and p , are used to select the distribution according to Eqs.(F.4) and (F.6). The parameters that describe the Johnson distribution are then calculated according to Tables F.1 and F.2.

Table F.1: Parameters for estimation of Johnson bounded and unbounded distribution.

Johnson unbounded S_U	Johnson bounded S_B
$z = \gamma + \eta \sinh^{-1} \left(\frac{x - \epsilon}{\lambda} \right)$	$z = \gamma + \eta \ln \left(\frac{x - \epsilon}{\lambda + \epsilon - x} \right)$
$\eta = \frac{2z}{\cosh^{-1} \left[\frac{1}{2} \left(\frac{m}{p} + \frac{n}{p} \right) \right]} ; \quad (\eta > 0)$	$\eta = \frac{z}{\cosh^{-1} \left(\frac{1}{2} \left[\left(1 + \frac{p}{m} \right) \left(1 + \frac{p}{n} \right) \right]^{1/2} \right)} ; \quad (\eta > 0)$
$\gamma = \eta \sinh^{-1} \left[\frac{\frac{n}{p} - \frac{m}{p}}{2 \left(\frac{m}{p} \frac{n}{p} - 1 \right)^{1/2}} \right]$	$\gamma = \eta \sinh^{-1} \left[\frac{\left(\frac{p}{n} - \frac{p}{m} \right) \left\{ \left(1 + \frac{p}{m} \right) \left(1 + \frac{p}{n} \right) - 4 \right\}^{1/2}}{2 \left(\frac{p}{m} \frac{p}{n} - 1 \right)} \right]$
$\lambda = \frac{2p \left(\frac{m}{p} \frac{n}{p} - 1 \right)^{1/2}}{\left(\frac{m}{p} + \frac{n}{p} - 2 \right) \left(\frac{m}{p} + \frac{n}{p} + 2 \right)^{1/2}} ; \quad (\lambda > 0)$	$\lambda = \frac{p \left[\left\{ \left(1 + \frac{p}{m} \right) \left(1 + \frac{p}{n} \right) - 2 \right\}^2 - 4 \right]^{1/2}}{\frac{p}{m} \frac{p}{n} - 1} ; \quad (\lambda > 0)$
$\epsilon = \frac{x_z + x_{-z}}{2} + \frac{p \left(\frac{n}{p} - \frac{m}{p} \right)}{2 \left(\frac{m}{p} + \frac{n}{p} - 2 \right)}$	$\epsilon = \frac{x_z + x_{-z}}{2} - \frac{\lambda}{2} + \frac{p \left(\frac{p}{n} - \frac{p}{m} \right)}{2 \left(\frac{p}{m} \frac{p}{n} - 1 \right)}$

The normal standard variable z can be used to construct the probability density function $P(z)$ (an example of which is shown in Fig. F.1) according to Eq.(F.7).

$$P(z) = \frac{e^{-\left(\frac{z^2}{2}\right)}}{\sqrt{2\pi}} \quad (\text{F.7})$$

Many simulation modelling packages have a facility to generate most of the common

Table F.2: Parameters for estimation of Johnson lognormal distribution.

$$\begin{array}{c}
 \hline
 \text{Johnson lognormal } S_L \\
 z = \gamma^* + \eta \ln(x - \epsilon) \\
 \hline
 \\
 \eta = \frac{2z}{\ln\left(\frac{m}{p}\right)} \\
 \\
 \gamma^* = \eta \ln \left[\frac{\frac{m}{p} - 1}{p \left(\frac{m}{p}\right)^{1/2}} \right] \\
 \\
 \epsilon = \frac{x_z + x_{-z}}{2} - \frac{p \frac{m}{p} + 1}{2 \frac{m}{p} - 1} \\
 \hline
 \end{array}$$

distributions given the distribution’s parameters. In particular, the four Johnson parameters ϵ , λ , γ and η can be used to reconstruct the Johnson curve and sample from it as necessary during the model runtime. A significant advantage of using this method is that it can be fully automated and the level of information transfer is very small, that is, rather than use all of the original empirical data, the four parameters can be used directly by the simulation package. A disadvantage to using this method is that the Johnson fit is used without any verification or ‘goodness of fit’ tests such as the Chi-Square Test or the Kilmogorov-Smirnov test, and therefore, there may be better distribution fits available that are not examined.

F.2 Software Interpretation of Johnson Distribution

Table F.3 shows how the Johnson distribution is interpreted by Statfit, ExtendSim and VB.

F.3 Python Implementation

The function `JohnsonFitter` takes a sequence of points from the data under observation returns a list of the four required parameters required to reconstruct the fitted Johnson distribution.

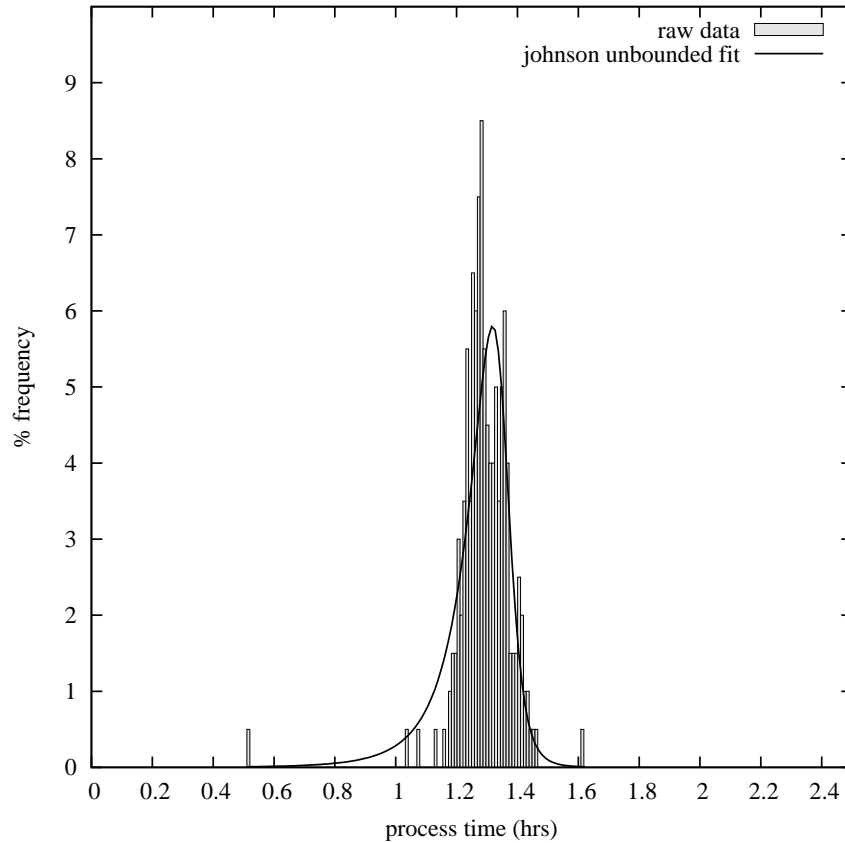


Figure F.1: An example of the empirical data and the fitted Johnson unbounded frequency distribution using the algorithm described in this chapter.

Table F.3: Different interpretation of Johnson distribution parameters.

Johnson distribution type	parameter	Statfit	ExtendSim	RandomCalculate Function (ExtendSim)	VB
Bounded S_B (ExtendSim dist. type 22)	minimum	min	location	Add to Random-Calculate()	ϵ
	range	δ	λ	Parameter 1	λ
	skewness	γ	γ	Parameter 2	γ
	shape	δ	δ	Parameter 3	η
Unbounded S_U (Extend-Sim dist. type 23)	minimum	min	location	Add to Random-Calculate()	ϵ
	range	δ	λ	Parameter 1	λ
	skewness	γ	γ	Parameter 2	γ
	shape	δ	δ	Parameter 3	η

```

from scipy.stats import norm, scoreatpercentile
from math import acosh, asinh, sqrt, pi, exp, log
from numpy import zeros

def JohnsonFitter(data, numBins=100):
    data.sort()
    sampleSize=len(data)
    if sampleSize<10:

```

```

print "insufficient data"
return -1
elif sampleSize>25:
    z=1.2
else:
    z=0.5

max_data=max(data)
min_data=min(data)
increment=(max_data-min_data)/(numBins)

bins=[]
for i in range(numBins*2): # ensure there's a high max
    bins.append(min_data+(i)*increment)
    if bins[i]>max_data: # terminates the loop
        break

quant=zeros(numBins)
for pt in data:
    for j in range(len(bins)-2):
        if pt>=bins[j] and pt<bins[j+1]:
            quant[j]=1
            break

sumQ=sum(quant)
for i in range(len(quant)):
    quant[i]=quant[i]/sumQ

plus_z=norm.cdf(z)
plus_3z=norm.cdf(3*z)
minus_z=norm.cdf(-1*z)
minus_3z=norm.cdf(-3*z)
plus_x=scoreatpercentile(data,plus_z*100)
plus_3x=scoreatpercentile(data,plus_3z*100)
minus_x=scoreatpercentile(data,minus_z*100)
minus_3x=scoreatpercentile(data,minus_3z*100)
m = plus_3x - plus_x
n = minus_x - minus_3x
p = plus_x - minus_x
ind=m*n/(p**2)

jVals=[]
if ind>1.1:
    "johnson UNbound"
    neta=2*z/acosh(0.5*(m/p + n/p))
    gamma=neta*asinh((n/p - m/p)/(2*((n/p)*(m/p)-1)**0.5))
    lamd=(2*p*((n/p)*(m/p)-1)**0.5)/((m/p - 2)*(m/p + n/p + 2)**0.5)
    epsilon=0.5*(plus_x+minus_x)+((p*(n/p - m/p))/(2*(m/p + n/p - 2)))
    for x in bins:
        jVals.append(exp((( -1)*((gamma+neta*asinh((x-epsilon)/lamd)**2)/2))/(sqrt(2*pi)))
elif ind<0.9:
    "johnson Bound"
    neta=z/acosh(0.5*((1+ p/n)*(1+p/m))**0.5)
    gamma=neta*asinh(((p/n - p/m)*((1+ p/m)*(1+p/n)-4)**0.5)/(2*((p/m)*(p/n)-1)))
    lamd=p*((1+ p/m)*(1+p/n)-2)**2 - 4)**0.5/((p/m)*(p/n)-1)
    epsilon=0.5*(plus_x+minus_x)-(lamd/2)+((p*(p/n - p/m))/(2*(p/m + p/n - 1)))
    for x in bins:
        jVals.append(exp((( -1)*((gamma+neta*log((x-epsilon)/(lamd+epsilon-x))**2)/2))/(sqrt(2*
pi))))
else: # 0.9 <= ind <= 1.1
    "johnson LOGnormal"
    "not implemented"
    neta=2*z/log(m/p)
    gamma=neta*log((m/p - 1)/(p*((m/p)**0.5)))
    epsilon=0.5*(plus_x+minus_x)-(p/2)*((m/p + 1)/(m/p - 1))
    for x in bins:
        jVals.append(exp((( -1)*((gamma+neta*log(x-epsilon))**2)/2))/(sqrt(2*pi)))

return neta,gamma,lamd,epsilon

```

F.4 VB Implementation

The Visual Basic (VB) implementation uses the `importData` to call either the `johnson_Bound` or the `johnson_Unbound` function which returns the parameters necessary to construct the proposed Johnson distribution.

```

Sub importData(data,inc)

    Dim NumDataPoints, i As Integer
    Dim MaxVal, minVal As Double
    Dim z, threeZ, minusThreeZ, minusZ As Double
    Dim Z_score, threeZ_score, minusThreeZ_score, minusZ_score As Double
    Dim DataPointsArray() As Double
    Dim xThree, xOne, xMinusOne, xMinusThree As Double
    Dim m, n, p As Double
    Dim mn_over_pSquared As Double
    Dim arraySize As Integer
    Dim stepperArray(), inc As Double
    Dim dataAverage As Double
    Dim temp As Double

    dataAverage = WorksheetFunction.average(data)
    minVal = WorksheetFunction.Min(data)
    maxVal = WorksheetFunction.Max(data)

    'Create Stepper array
    arraySize = 100
    ReDim stepperArray(arraySize)

    temp = 0
    Do Until temp > minVal
        temp = temp + inc
    Loop

    stepperArray(1) = temp - inc

    i = 1
    Do
        i = i + 1
        If i >= arraySize Then
            ReDim Preserve stepperArray(i + arraySize)
            arraySize = i + arraySize
        End If
        stepperArray(i) = stepperArray(i - 1) + inc
    Loop Until stepperArray(i) >= MaxVal

    ReDim Preserve stepperArray(i)
    arraySize = i

    'Get z values and z scores
    If NumDataPoints >= 25 Then
        z = 1.2
    ElseIf NumDataPoints >= 10 Then
        z = 0.5
    Else
        MsgBox ("No Bin Width Specified")
        Exit Sub
    End If

    minusZ = z * (-1)
    threeZ = 3 * z
    minusThreeZ = threeZ * (-1)
    Z_score = WorksheetFunction.NormSDist(z)
    minusZ_score = WorksheetFunction.NormSDist(minusZ)
    threeZ_score = WorksheetFunction.NormSDist(threeZ)
    minusThreeZ_score = WorksheetFunction.NormSDist(minusThreeZ)

    'Get Percentile
    xThree = WorksheetFunction.Percentile(DataPointsArray, threeZ_score)
    xOne = WorksheetFunction.Percentile(DataPointsArray, Z_score)
    xMinusOne = WorksheetFunction.Percentile(DataPointsArray, minusZ_score)
    xMinusThree = WorksheetFunction.Percentile(DataPointsArray, minusThreeZ_score)

```


Appendix F. Johnson Distribution

```

'Get m, n and p values
m = xThree - xOne
n = xMinusOne - xMinusThree
p = xOne - xMinusOne

'Select Distribution
mn_over_pSquared = (m * n) / (p * p)
Select Case mn_over_pSquared
  Case Is > 1
    Call johnson_Unbound(m, n, p, z, xOne, xMinusOne, stepperArray, arraySize, DataPointsArray)
  Case Is < 1
    Call johnson_Bound(m, n, p, z, xOne, xMinusOne, stepperArray, arraySize, DataPointsArray)
End Select
End Sub

```

```

Sub johnson_Unbound(m, n, p, z, xOne, xMinusOne, stepperArray, arraySize, DataPointsArray)

  Dim neta, gamma, lambda, epsilon, sumArray As Double
  Dim tempValue, johnsonSuArray() As Double
  Dim i As Integer

  neta = (2 * z) / (WorksheetFunction.Acosh(0.5 * ((m / p) + (n / p))))
  gamma = neta * WorksheetFunction.Asinh(((n / p) - (m / p)) / (2 * (((m / p) * (n / p)) - 1) ^ 0.5)))
  lambda = ((2 * p) * (((m / p) * (n / p)) - 1) ^ 0.5)) / ((m / p) + (n / p) - 2) * Sqr((m / p) + (n / p) + 2)))
  epsilon = ((xOne + xMinusOne) * 0.5) + (p * ((n / p) - (m / p))) / (2 * ((m / p) + (n / p) - 2))

  ReDim johnsonSuArray(arraySize)

  For i = 1 To arraySize
    tempValue = gamma + (neta * WorksheetFunction.Asinh((stepperArray(i) - epsilon) / lambda))
    johnsonSuArray(i) = Exp((-1) * (tempValue ^ 2) / 2)) / (Sqr(2 * WorksheetFunction.Pi))
  Next i

  sumArray = 0
  For i = 1 To arraySize
    sumArray = sumArray + johnsonSuArray(i)
  Next i

  For i = 1 To arraySize
    johnsonSuArray(i) = johnsonSuArray(i) / sumArray
  Next i

  return johnsonSuArray
End Sub

```

```

Sub johnson_Bound(m, n, p, z, xOne, xMinusOne, stepperArray, arraySize, DataPointsArray)

  Dim neta, gamma, lambda, epsilon As Double
  Dim tempValue, sumArray, johnsonSbArray() As Double
  Dim i As Integer

  neta = z / (WorksheetFunction.Acosh(0.5 * (((1 + (p / m)) * (1 + (p / n))) ^ 0.5)))
  gamma = neta * WorksheetFunction.Asinh((((p / n) - (p / m)) * (((1 + (p / m)) * (1 + (p / n) - 4) ^ 0.5)) / (2 * (((p / m) * (p / n)) - 1))))
  lambda = p * ((((((1 + (p / m)) * (1 + (p / n))) - 2) ^ 2) - 4) ^ 0.5) / (((p / m) * (p / n)) - 1))
  epsilon = ((xOne + xMinusOne) * 0.5) - (lambda / 2) + (p * ((p / n) - (p / m))) / (2 * (((p / m) * (p / n)) - 1))

  ReDim johnsonSbArray(arraySize)
  For i = 1 To arraySize
    If ((stepperArray(i) - epsilon) / (lambda + epsilon - stepperArray(i))) > 0 Then
      tempValue = gamma + (neta * WorksheetFunction.Ln((stepperArray(i) - epsilon) / (lambda + epsilon - stepperArray(i))))
      johnsonSbArray(i) = Exp((-1) * (tempValue ^ 2) / 2)) / (Sqr(2 * WorksheetFunction.Pi))
    End If
  Next i

  sumArray = 0
  For i = 1 To arraySize

```

```
    sumArray = sumArray + johnsonSbArray(i)
Next i
For i = 1 To arraySize
    johnsonSbArray(i) = johnsonSbArray(i) / sumArray
Next i
return johnsonSbArray
End Sub
```

Verification and Validation Techniques

G.1 Verification Techniques

Verification is concerned with determining whether a conceptual model of a real system has been correctly translated into a computer program (Banks and Gibson, 1997b; Law and Kelton, 1997; Law, 2008; Robinson, 1997; Sargent, 1998). Table G.1 contains a comprehensive list of techniques used to verify that a computer model is a good representation of the conceptual model. The list was produced by Whitner and Balci (1989) and categorised under six main headings,

Informal Analysis through informal consultations and activities. Many of the techniques involve human intuitive decisions and reasoning.

Static Analysing the computer source code that constitutes the model. Static analysis

does not involve execution of the model. In fact, the code compiler is a form of static analysis.

Dynamic Analysing computer model parameters during model runtime. Many programmers refer to this step as debugging.

Symbolic Analysing transformation of inputs to outputs during model runtime.

Constraint Analysing the comparison between state transformation in the model with the list of assumptions that the conceptual model is based on.

Formal Analysis through formal mathematical and scientific procedures.

Table G.1: Verification techniques for simulation modelling (Whitner and Balci, 1989).

Type	Technique	Description
Informal	Desk Checking	Checking the logic, consistency and completeness of the model. Usually performed prior to debugging and execution.
	Walk-through	The walk-through is similar to desk checking but is usually performed by all the parties with a stake in the success of the simulation.
	Code Inspection Review	Carrying out a line by line inspection of the source code. A review check that the intended specification standards and guidelines of the model have been adhered to in the final version.
	Audit	The audit is more concerned with the success of the development process.
Static	Syntax Analysis	Ensures that the syntax of the computer language used is correct. This part is usually carried out by the compiler and unnecessary in most modern simulation software.
	Semantic Analysis	Confirmation that the computer language functions and syntax are in line with the actions the modeller intended them to perform.
	Structural Analysis	Testing the model structure for breaches of basic language structural errors.
	Data Flow Analysis	Tracing through the model from the perspective of a single thread to ensure all possible eventualities have been included.
	Consistency Analysis	Ensuring that the model does not contain any contradictions and model variable are consistent in their type and use throughout the model.
Dynamic	Top-down Testing	Top down testing involves testing each subdivision from the top most to the bottom of the model using dummy inputs.
	Bottom-up Testing	Bottom up testing is similar to top-down testing but involves beginning at the lowest sub-model.

continued on next page

Appendix G. Verification and Validation Techniques

continued from last page

	Black-box testing	Involves testing for any inputs that produce erroneous outputs.
	White-box Testing	As opposed to black-box testing, white-box testing is concerned with the internals of the model and the data flow through the code.
	Stress Testing	Stress testing involves checking the system with inputs that are excessively beyond the normal input ranges.
	Debugging	Debugging is the actual process of removing errors from the model. It is not really testing more the correction based on results from testing.
	Execution Tracing	Execution tracing involves watching a number of parameters or variables in the model develop over the course of its execution.
	Execution Monitoring	Monitoring the actual activities that the model is performing during execution. This could involve a trace animation on a graphical simulation package.
	Execution Profiling	This is very similar to execution monitoring but done on a more macro level. Group movements and action are mainly monitored here.
	Symbolic Debugging	Symbolic debugging is a method of debugging whereby the modeller may examine various variables, change them during execution or replay certain segments of the model execution.
	Regression Testing	Regression testing ensures that model corrections or development steps to not cause new issues or error. It involves retesting the updated model with a number of the same tests that were performed before it was updated.
Symbolic	Symbolic Execution	The models symbolic values are examined as opposed to the actual program values. The result of this examination can often be shown in a symbolic decision making tree.
	Path Analysis	Path Analysis involves testing every possible route or path through the model for potential errors and completeness.
	Cause-Effect Graphing	This involves documenting a large graph of the relationship between the causes and effects of the simulation model
	Partition Analysis	Partition analysis is done by subdividing the model into its component sub-models and testing each individual sub-model against its correct intended use according to specifications.
Constraint	Assertion Checking	Assertion checking involves ensuring that certain rules about how the model operates are never broken. For example, if a particular variable or parameter is always non-negative than coded testing loops can be employed to check that the variable of interest is always non-negative.
	Inductive Assertion	The input/output relations are translated into assertions about the transition. If the model can be traversed without any assertions being broken then the model could be deemed as verified.
	Boundary Analysis	Assuming that the most error prone test cases lie on the boundaries of the model operating ranges, this analysis runs test data at or around the boundaries.
Formal	Proof of Correctness	Use the technique of a formal logic system to prove that if the input values satisfy certain constraints, the output values produced by the program, satisfy certain properties.

continued on next page

continued from last page

Lambda Calculus	The model is transferred into formal expressions using string functions. The model itself can be considered a large string. Lambda calculus specifies rules for rewriting strings to transform the model into lambda calculus expressions. Using lambda calculus, the modeller can express the model formally to apply mathematical proof of correctness techniques to it (Barendregt, 1984).
Predicate Calculus	A predicate is a combination of simple relations that are given boolean values of either true or false. The model can be defined in terms of predicates and manipulated using the rules of predicate calculus (Backhouse, 1986).
Predicate Transformation	Predicate transformation involves formally defining the semantics of the model with a mapping that transforms model output states to all possible model input states (Youngblood, 2006).
Induction, Inference, and Logical Deduction	These tasks validate the model by justifying conclusions on the basis of the specification premises. Arguments are deemed valid if the path from premise to conclusion conforms to the established rules of inference (Youngblood, 2006).

G.2 Validation Techniques

Validating a model “refers to the processes and techniques that the model developer, model customer and decision makers jointly use to assure that the model represents the real system (or proposed real system) to a sufficient level of accuracy” (Carson, 2002). Law and Kelton (1997) define validation as the “process of determining whether a simulation model ... is an accurate representation of the system, for the particular objectives of the study”. Table G.2 gives a comprehensive list, compiled by Sargent (1998), of the validation techniques discussed in the area of simulation.

Table G.2: Validation techniques for simulation models (Sargent, 1998).

Technique	Description
Animation	Validation and debugging using animation has become more accessible with the advent of graphical simulation packages. Often both symbolic animation and trace animation are available on modern simulation packages.
Comparison to other models	Using results from other previously validated models to compare with.
Degenerate Tests	Checking that the model results become increasingly unstable when the model is tested outside of its normal operational bounds.
Event Validity	The discrete events of the simulation model are consistent with the series of events that could take place in the real world system.

continued on next page

continued from last page

Extreme Conditions Test	Similar to Degenerate tests, e.g., extremities such as zero throughput and zero queueing should coexist.
Face Validity	Face validity involves interviewing and questioning people that are well versed with the real system, it is perhaps one of the most common methods but is hard to quantify and not always documented particularly well.
Fixed Values	In some models it is possible to estimate an output value for a particular model configuration and input values. The constants should be checked for consistency.
Historical Data Validation	Historical data records are a very powerful tool for validating a model. Model output data driven with empirical historical records can be compared to the output.
Historical Methods	Include methods such as positive economics, rationalism and empiricism (see Sargent (1998) for description).
Internal Validity	The model is tested for output variability of results, which may indicate model instability.
Operational Graphics	Monitoring system variables using animation.
Parameter Variability-Sensitivity Analysis	Testing and comparing model inputs with model outputs.
Predictive Validation	Using the model to predict future real system performance and comparing it to actual real system future performance.
Traces	Tracing model entities as they flow through the model. This ensures the internal logic of the model is reasonable.
Turing Tests	Performing blind tests on the data outputted from both the model and the real system.

Publications

The following is a list of publications arising from this thesis. The list is in chronological order with the most recent first. It includes conference papers, poster presentations as well as forthcoming publications and work stemming from the topics discussed.

Byrne, N. M., Liston, P., Geraghty, J. and Young, P. 2012. Open Source Discrete Event Simulation Software, *Proceedings of the Operational Research Society Simulation Workshop 2012*, [Approved].

Liston, P., Byrne, N. M., Geraghty, J. and Young, P. 2011. Collaborating to Gain Advantage from Non-Core Manufacturing Competencies, *Proceedings of the 28th International Manufacturing Conference*, pp. 136-144.

Byrne, N. M., Liston, P., Geraghty, J., Young, P. and O' Donovan, A. 2011. An Industry View of Discrete Event Simulation, *Proceedings of the Intel European Research and Innovation Conference 2011*.

Byrne, N. M., Geraghty, J. and Young, P. 2010. Generating Operating Curves for Semi-

- conductor Toolsets, *Proceedings of the Intel European Research and Innovation Conference 2010*.
- Byrne, N. M., Geraghty, J., Young, P., Sievwright, S. and Daly, K. J. 2009. Generating Operating Curves for Semiconductor Toolsets, *Proceedings of the Intel European Research and Innovation Conference 2009*.
- Byrne, N. M., Geraghty, J., Young, P., Sievwright, S. and Daly, K. J. 2008. Operational Characteristic Curves for Semiconductor Fabs. *Proceedings of the Intel European Research and Innovation Conference 2008*.
- Byrne, N. M., Geraghty, J., Young, P., Sievwright, S. and Daly, K. J. 2007. Methodology for filtering real data into distributions for use in complex manufacturing simulation. *Proceedings of the 24th International Manufacturing Conference*, 2, pp. 883-892.
- Byrne, N. M., Geraghty, J., Young, P., Sievwright, S. and Daly, K. J. 2007. Methodology for Complex Semiconductor Manufacturing for Discrete Event Simulation Projects. *Proceedings of the Intel European Research and Innovation Conference 2007*.
- Byrne, N. M., Geraghty, J., Young, P., Sievwright, S. and Daly, K. J. 2007. An intelligent queueing block for simulation software. *Proceedings of the 3rd Workshop on Simulation in Manufacturing, Services and Logistics*.