The Dynamics of Project Management: An Investigation of the Impacts of Project Process and Coordination on Performance

by

David N. Ford

Master of Engineering, Tulane University, New Orleans (1979)
B.S. Civil Engineering, Tulane University, New Orleans (1976)

Submitted to the Department of Civil and Environmental Engineering in Partial Fulfillment of the Requirements for the Degree of

Doctor of Philosophy
in
Dynamic Engineering Systems

at the
Massachusetts Institute of Technology

August, 1995

©1995 Massachusetts Institute of Technology. All rights reserved.

Signature of Author

David N. Ford

Department of Civil and Environmental Engineering
August, 15, 1995

Certified by

John D. Sterman
Professor of Management Science
Research Advisor

Certified by

Jerome J. Connor
Professor of Civil and Environmental Engineering
Committee Chairman

Accepted by

Joseph M. Sussman
Chairman, Departmental Committee on Graduate Students

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

OCT 25 1995 ARCHIVES

LIBRARIES
The Dynamics of Project Management: An Investigation of the Impacts of Project Process and Coordination on Performance

by

David N. Ford

Submitted to the Department of Civil and Environmental Engineering on August 15, 1995 in Partial Fulfillment of the Requirements for the Degree of Doctor of Philosophy in Dynamic Engineering Systems

Abstract

Successful product development projects are critical to competitiveness in several industries. Successful management of these projects requires an understanding and use of the dynamics of projects. Existing research has focused on a static view of project management. This research investigates the impacts of dynamic project structure on performance with a focus on the influence of the development process.

A dynamic simulation model of a multiple phase project was built using the system dynamics methodology. The model integrates several previously developed and tested project structures and adds a separate structure for the development process. Simulations describe the behavior generated by the interaction of customized development project phases and a project management structure. Project performance is measured in time, quality and cost. The model was calibrated to a computer chip development project for a single development phase configuration and a four phase configuration which represented the majority of the development process. Testing revealed that when the model is appropriately parameterized the resulting simulated behavior closely resembles the historical behavior of the project.

The model was applied to the investigation of coordination policies for improved project performance. Analysis of the influences of two descriptors of coordination policy reveal that cycle time can decrease as the delay between coordination labor need and
coordination labor provided increases. The model structure helped identify the timing of a shift in feedback loop dominance as the cause of this counterintuitive behavior.

The research finds that development processes significantly impact the dynamic behavior of projects through the feedback, delays and nonlinear relationships which are not used in traditional project models but are important descriptors of project complexity. Expanding the models used to manage projects to include dynamic features requires a change of focus by researchers and practitioners. The system dynamics methodology provides some of the tools for developing and implementing such an expanded project model. Future research using the model within and beyond its current limits can facilitate the development of new knowledge of project dynamics and the implementation of that knowledge in project management practice.

**Thesis Committee**

Dr. John D. Sterman, Research Supervisor  
Professor of Management Science  
Sloan School of Management

Dr. Jerome J. Connor, Committee Chairman  
Professor of Civil and Environmental Engineering

Dr. Stephen Eppinger  
Associate Professor of Management Science  
Sloan School of Management

Dr. John Williams  
Associate Professor of Civil and Environmental Engineering
This dissertation is dedicated

to my father

Griffin Thomas Nelson (1926 - 1956)

and my grandfather

Sylvester Henry Nelson (1895 - 1980)
Acknowledgments

Many people have contributed to the work which culminates in this thesis. Primary among them are my doctoral committee. Professor Sterman taught me by example the potential contributions of practicing excellence in research and teaching. As the chairman of my committee Professor Connor shepherded the final miles of my journey through the Civil and Environmental Engineering Department and initially directed me to the system dynamics work at Sloan. Professor Eppinger unfailingly provided insightful, truly helpful critiques and advise, support, and a treasured glimpse of the adventures of a junior faculty member. Professor Williams urged me in his own unique way to improve my research. These men have largely formed my image of the professor-doctoral student relationship. They have each given me models from which I will create my own style of guiding my own students.

Others at MIT also impacted my academic life and work. Professor Logcher strongly influenced both the direction of my work during and my experience of my initial years at MIT. Jim Becker allowed me to participate in teaching and provided an essential link to the realities of life in the construction industry. Henry Irwig was unfailingly supportive and always sincerely interested in my work. Some of my most enjoyable research memories are of the project on which we collaborated. The late Professor James Paddock was amazingly helpful in the brief time he was on my General Exam committee. Funding as a PACT fellow and by Sloan's Organizational Learning Center are appreciated. I am grateful for my time with a host of graduate students and MIT staff including my numerous office mates in Building 1, Daniel and Rory, MIT's Center for Real Estate staff, the System Dynamics Group, and the Organizational Learning Center.

My work on my dissertation and the research which preceded it benefited from the excellent tutelage and guidance of Professor Sterman, the extensive efforts of Terry Johnson and the rest of the Genesis team at International Chip Inc. and the flexibility of Janet Gould in her leadership of the OLC's project with ICI. Special thanks are due to Roger Oliva and Daniel Higgins for their support and special help during the final sprint toward the defense. My wife Laura's critical eye for the visually pleasing as well as her editing and extensive formatting efforts greatly improved the final document.

I owe thanks to those who have helped me maintain a life within which to work during my time at MIT. Primary among them are my wife Laura and Billy and Charlotte, who created and kept a home for me to come home to. Laura and I surviving the first year of our marriage simultaneously with my final year at MIT can only bode well for the rest of our lives together. Finding and keeping the elusive healthy perspective was also greatly facilitated by dinners with Rory, coffee and philosophy with Daniel, the weekly Table Function at the Marriot, the Friday lunch group, and the gothic architecture of Emmanuel Church of Boston.

Finally, thanks are due to two dear friends, one whom I am separated from only by time and the other only by space. To Minew, who never understood why we left Texas and only wanted what she deserved: "I'm done Sweety. We can go home now." And to Beverley, who was there so many years ago when I was writing my Master's thesis and swearing I'd never go to graduate school again: "This time I mean it, really."
Table of Contents

List of Figures ................................................................. 15

List of Tables ............................................................... 21

CHAPTERS

1 Introduction ................................................................. 23
  1.1 Context ........................................................................... 23
  1.2 Motivation for Research ............................................... 27
  1.3 The Problem ..................................................................... 29
  1.4 Research Approach ...................................................... 34
  1.5 Summary ......................................................................... 36

2 Literature Review ........................................................... 37
  2.1 Introduction ...................................................................... 37
  2.2 Traditional Characterizations of the Product Development Process .......... 37
  2.3 Recent Characterizations of the Product Development Process .................. 42
    2.3.1 Concurrent Development ................................... 42
    2.3.2 Cross-Functional Development Teams ......................... 45
  2.4 Summary of the Product Development Process Literature Review ............... 47
  2.5 System Dynamics Literature Review ................................... 48
2.5.1 Six Key Feedback Structures in System Dynamics Models of Projects ........................................ 48
  2.5.1.1 The Labor Structure ........................................ 48
  2.5.1.2 The Schedule Structure ..................................... 49
  2.5.1.3 The Rework Structure ....................................... 50
  2.5.1.4 The Available Work Structure ............................ 52
  2.5.1.5 The Quality Structure ...................................... 54
  2.5.1.6 The Scope Structure ....................................... 56

2.5.2 System Dynamics Models of Product Development Projects .......... 57

2.5.3 Evaluation of System Dynamics Models of Product Development Projects ........................................ 60

2.6 Summary of Literature Evaluation .................................................. 61

3 Model Description 63

3.1 Introduction ........................................................................... 63

3.2 Model Boundary and Level of Aggregation ..................................... 64

3.3 Model Structure ................................................................ 67
  3.3.1 Introduction ............................................................. 67
  3.3.2 Model Subsystems ..................................................... 68
  3.3.3 The Process Structure and Scope Systems .......................... 69
    3.3.3.1 Introduction ..................................................... 69
    3.3.3.2 The Development Task Sector ............................... 69
    3.3.3.3 The Errors Sectors .......................................... 83
  3.3.4 The Resources Subsystem .............................................. 91
    3.3.4.1 The Labor Sectors ........................................... 92
    3.3.4.2 The Experience Sector ...................................... 98
    3.3.4.3 The Quality of Practice Sector ............................. 100
    3.3.4.4 The Expected Productivity Sectors ....................... 103
Table of Contents

3.3.5 The Targets and Performance Subsystems ........................................... 107
   3.3.5.1 The Schedule Sector ...................................................... 107
   3.3.5.2 The Project Quality Sector ............................................. 111
   3.3.5.3 The Cost Sector ........................................................... 113
3.3.6 Basis for Key Model Structures ..................................................... 115
3.4 Model Behavior ................................................................................. 117
   3.4.1 Introduction ........................................................................... 117
   3.4.2 Typical Single Phase Model Behavior ....................................... 117
   3.4.3 Typical Multiple Phase Model Behavior .................................... 120
3.5 Model Sensitivity to Parameter Values ................................................ 123
   3.5.1 Introduction ........................................................................... 123
   3.5.2 Parameter Reduction .............................................................. 123
   3.5.3 Single Phase Model Sensitivity ............................................... 124
   3.5.4 Multiple Phase Model Sensitivity ........................................... 129
   3.5.5 Model Sensitivity Test Summary .............................................. 134
3.6 Model Description Summary ............................................................... 135

4 A Signal Processing Model of a Product Development Process Phase ........ 137
4.1 Introduction ....................................................................................... 137
4.2 System Descriptions ....................................................................... 138
4.3 System Behavior ............................................................................... 142
   4.3.1 Introduction ........................................................................... 142
   4.3.2 Conditions for Stability .......................................................... 142
   4.3.3 System Changes Which Impact Stability ................................... 148
   4.3.4 Conditions for System Induced Oscillation ................................ 148
   4.3.5 Typical System Behavior ......................................................... 149
      4.3.5.1 System Response to Impulse Signal .................................... 149
      4.3.5.2 System Response to Block Signal ...................................... 152
Table of Contents

4.4 Discussion of the Signal Processing Modeling Approach ....................... 154
  4.4.1 Alternative System Descriptions and Analysis Tools ..................... 154
  4.4.2 Applicability of Linearity Assumptions .................................... 155
  4.4.3 Tractability of Analyzing Larger Systems ................................. 156
4.5 Signal Processing Modeling Approach Summary ................................. 156

5 The Python Development Project .................................................. 159
  5.1 Introduction ............................................................................. 159
  5.2 Data Collection ....................................................................... 160
  5.3 Computer Chip Development at ICI .......................................... 162
    5.3.1 A Generic Computer Chip Development Process ....................... 162
    5.3.2 The Python Development Project Context ............................... 163
    5.3.3 The Development Process .................................................. 163
    5.3.4 Development Process Products ........................................... 168
    5.3.5 Product Development Project Metrics ................................... 169
    5.3.6 The Development Organization ............................................. 170
  5.4 The Python Development Project ................................................. 170
5.5 Model Calibration ...................................................................... 173
  5.5.1 Model Structure and the Python Development Project ................. 173
    5.5.1.1 Overview ...................................................................... 173
    5.5.1.2 Model Boundary Test .................................................... 175
    5.5.1.3 Customization of the Product Development Project Model Structure to Reflect the Python Development Project .......... 181
  5.5.2 Parameter Estimation .......................................................... 185
    5.5.2.1 One Phase Model Parameter Estimates ................................ 186
    5.5.2.2 Multiple Phase Model Parameter Estimates .......................... 197
  5.5.3 Comparison of Model Simulations to Python Project Behavior ....... 207
5.5.3.1 Single Phase Model Behavior ........................................... 207
5.5.3.2 Causes of Single Phase Model Behavior ......................... 214
5.5.3.3 Multiple Phase Model Behavior ...................................... 221
5.5.3.4 Causes of Multiple Phase Model Behavior ....................... 225
5.5.4 Model Calibration Summary ............................................... 229
5.6 Policy Analysis ................................................................... 229
5.6.1 Literature Review of Product Development
Project Coordination ................................................................ 230
5.6.1.1 Traditional Characterizations of
Coordination of Product Development Projects ....................... 230
5.6.1.2 Recent Characterizations of
Coordination of Product Development Projects ....................... 231
5.6.1.3 Coordination Tools, Expected Influences
and Benefits ........................................................................... 232
5.6.2 Model Representation of Coordination Policies .................... 235
5.6.2.1 Coordination Parameters ............................................... 235
5.6.2.2 Three Coordination Policies ......................................... 236
5.6.3 Project Performance Under Different Coordination Policies ... 237
5.6.4 Analysis of Project Behavior
in Response to Coordination Policies ....................................... 240
5.6.5 Causes of Project Behavior
in Response to Coordination Policies ....................................... 244
5.6.6 Impacts of Coordination Policy Analysis
on Coordination Policy Design ................................................. 250
5.6.7 Policy Analysis Summary .................................................. 253
5.7 The Python Development Project Summary ........................... 254

6 Conclusions ........................................................................... 255
6.1 Recapitulation ...................................................................... 255
6.2 Major Findings and Discussion ............................................. 257
6.2.1 The Role of Development Process in Project Behavior .......... 257
Table of Contents

6.2.2 The Role The Role of Feedback, Delays and Nonlinear Relationships Project Behavior .......................... 258
6.2.3 Project Constraints ................................................................. 258
6.2.4 An Important Gap in Project Management .................................... 259
6.2.5 System Dynamics as a Tool for Researching Projects ..................... 260

6.3 Implications for Project Management ............................................ 261
6.4 Contributions .................................................................................. 263
6.4.1 A New Research Tool ............................................................... 263
6.4.2 Project Dynamics Insights and A Tool for Product Development Practitioners ............................................. 264

6.5 Limitations of the Research .......................................................... 265
6.6 Future Research .............................................................................. 267
6.7 Summary Conclusions .................................................................... 267

References ......................................................................................... 269

Appendices

3.1 Model Equations ............................................................................ 278
3.2 Model Variables .............................................................................. 305
4.1 Signal Processing Model ................................................................. 328
5.1 Parameter Estimates for Model Calibration ..................................... 333
List of Figures

1-1 A Development Project Feedback Structure .................................................. 30
1-2 An Unintended Side Effect of a Project Management Policy ........................... 31
1-3 A Delay and A,Nonlinear Relationship in a Project System ........................ 32

2-1 A Network Diagram for Critical Path Method Schedule Management .......... 40
2-2 A Gantt Chart Representation of a Critical Path Method Schedule ................. 40
2-3 Sequential and Concurrent Product Development Processes ......................... 43
2-4 The Labor Structure .................................................................................. 48
2-5 The Schedule Structure ............................................................................ 50
2-6 The Rework Structure ............................................................................. 51
2-7 The Available Work Structure .................................................................. 53
2-8 The Quality Structure .............................................................................. 55
2-9 The Scope Structure ................................................................................ 56

3-1 Product Development Project Model Boundary Diagram ............................. 65
3-2 A Phase Dependency Network for a Development Project .......................... 68
3-3 Model Subsystems .................................................................................... 69
3-4 A Development Process Feedback Structure .............................................. 71
3-5 A Linear Internal Precedence Relationship ............................................... 72
3-6 A Nonlinear Internal Precedence Relationship .......................................... 73
List of Figures

3-7 Process Structure Stocks and Flows (Single Phase) ...................................................... 74
3-8 Process Structure (Single Phase) .................................................................................. 75
3-9 A "Lockstep" Constraint Described with An External Precedence Relationship ...................... 81
3-10 A Delayed Concurrent Constraint Described with An External Precedence Relationship .............. 82
3-11 The Development Tasks Sector ..................................................................................... 83
3-12 The Internal Errors Sector ............................................................................................ 84
3-13 The Upstream Errors Sector .......................................................................................... 88
3-14 The Gross Labor Sector .................................................................................................. 92
3-15 The Labor Allocation Sector ............................................................................................ 94
3-16 The Workweek Sector .................................................................................................... 97
3-17 The Experience Sector .................................................................................................... 99
3-18 The Quality of Practice Sector ....................................................................................... 101
3-19 The Expected Basework Productivity Sector ..................................................................... 103
3-20 The Expected Quality Assurance Productivity Sector ....................................................... 103
3-21 The Expected Rework Productivity Sector ....................................................................... 104
3-22 The Expected Coordination Productivity Sector .................................................................. 104
3-23 The Project Deadline Sector ........................................................................................... 108
3-24 The Project Quality Sector .............................................................................................. 111
3-25 Single Phase Simulation with No Errors Development Task Sector Flows ............................... 118
3-26 Single Phase Simulation with No Errors Development Task Sector Stocks ............................. 118
3-27 Single Phase Simulation with 50% Error Generation Development Task Sector Flows .................. 119
3-28 Single Phase Simulation with 50% Error Generation Development Task Sector Stocks .................. 119
3-29 Upstream Simulation of Two Phase Model Development Task Sector Flows .............................. 120
<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>3-30</td>
<td>Upstream Simulation of Two Phase Model Development Task Sector Stocks</td>
</tr>
<tr>
<td>3-31</td>
<td>Downstream Simulation of Two Phase Model Development Task Sector Flows</td>
</tr>
<tr>
<td>3-32</td>
<td>Downstream Simulation of Two Phase Model Development Task Sector Stocks</td>
</tr>
<tr>
<td>3-33</td>
<td>Multiple Phase Model for Sensitivity Testing</td>
</tr>
<tr>
<td>4-1</td>
<td>A System Dynamics Representation of a Product Development Process Phase Model</td>
</tr>
<tr>
<td>4-2</td>
<td>A Signal Processing Representation of a Product Development Process Phase Model</td>
</tr>
<tr>
<td>4-3</td>
<td>The System Poles Plotted in the S-Plane</td>
</tr>
<tr>
<td>4-4</td>
<td>System Response (Tasks Released) to Pulse Signal with T1 = -10</td>
</tr>
<tr>
<td>4-5</td>
<td>System Response (Task Released) to Impulse Signal</td>
</tr>
<tr>
<td>4-6</td>
<td>System Response (Known Rework) to Impulse Signal</td>
</tr>
<tr>
<td>4-7</td>
<td>System Response (Task Completed but not Checked) to Impulse Signal</td>
</tr>
<tr>
<td>4-8</td>
<td>System Response (Task Release Rate) to Impulse Signal</td>
</tr>
<tr>
<td>4-9</td>
<td>System Response (Task Release Rate) to Block Signal Impact of Inspection Time Constant</td>
</tr>
<tr>
<td>4-10</td>
<td>System Response (Task Release Rate) to Block Signal Impact of Inspection Failure Fraction (IFF)</td>
</tr>
<tr>
<td>5-1</td>
<td>ICI's Stage-Gate Development Process</td>
</tr>
<tr>
<td>5-2</td>
<td>Project Phase Network for the Python Development Project</td>
</tr>
<tr>
<td>5-3</td>
<td>Primary Feedback Structures Identified at Model Boundary Workshop</td>
</tr>
<tr>
<td>5-4</td>
<td>The Additional Stock Required to Model the Aggregate Release of Tasks from a Phase</td>
</tr>
<tr>
<td>5-5</td>
<td>Additional Model Structure Representing the Aggregation and Holding of Development Tasks before Release</td>
</tr>
<tr>
<td>5-6</td>
<td>Design Phase Internal Precedence Relationship for Calibration</td>
</tr>
</tbody>
</table>
5-7 Cumulative Basework and Error Checking
for QA Labor Delay Parameter Estimate ................................................. 197

5-8 Critical Path Method Estimate of Static Delay
between Design and Prototype Test Phases: Activity Chart .............. 206

5-9 Critical Path Method Estimate of Static Delay
between Design and Prototype Test Phases: Activity Timeline .......... 206

5-10 Design Phase Model Calibration to Historical Behavior
Model Parameters with Historical Data Available ........................................ 208

5-11 Design Phase Model Calibration to Historical Behavior
Basework Rate .................................................................................. 209

5-12 Design Phase Model Calibration to Historical Behavior .......... 211

5-13 Design Phase Model Calibration to Historical Behavior
Rework Rate ..................................................................................... 212

5-14 Design Phase Model Calibration to Historical Behavior
Tasks Completed but not Checked and Tasks Held for Release .......... 213

5-15 Single Phase Available Work Feedback Structure ................. 214

5-16 Single Phase Labor Availability Feedback Structure ............... 216

5-17 Single Phase Error Feedback Structure ....................................... 217

5-18 Compensating Feedback Loops in Schedule Project Target Structure .... 219

5-19 Compensating Feedback Loops in Quality Project Target Structure .... 220

5-20 Multiple Phase Project Simulation .................................................. 221

5-21 Multiple Phase Model Calibration to Historical Behavior
Product Definition Tasks Released ....................................................... 222

5-22 Multiple Phase Model Calibration to Historical Behavior
Design Tasks Completed and Held for Release ................................. 223

5-23 Multiple Phase Model Calibration to Historical Behavior
Design Tasks Released ..................................................................... 223

5-24 Multiple Phase Model Calibration to Historical Behavior
Prototype Testing Tasks Released ....................................................... 224

5-25 Multiple Phase Model Calibration to Historical Behavior
Reliability/Quality Control Tasks Released ......................................... 224

5-26 Inter-Phase Available Work Feedback ........................................... 226
List of Figures

5-27  Multiple Phase Model Error Flows ................................................................. 227
5-28  Project Simulation
      No Coordination Policy Project ........................................................................ 238
5-29  Project Simulation
      Passive Coordination Policy Project ................................................................ 238
5-30  Project Simulation
      Active Coordination Policy Project ................................................................ 239
5-31  Project Quality Performance Versus Coordination Labor Delay
      and Coordination Policy ...................................................................................... 242
5-32  Project Cycle Time Performance Versus Coordination Labor Delay
      and Coordination Policy ...................................................................................... 243
5-33  Causal Relationships Linking Coordination Labor Delay
      and Coordination Priority to Project Quality ....................................................... 245
5-34  Causal Relationships Linking Coordination Priority
      to Project Cycle Time ............................................................................................. 246
5-35  Causal Links from Coordination Labor Delay to Cycle Time ....................... 247
5-36  Coordination Policy Hypothesis Test ................................................................. 250
5-37  Project Simulation of A Coordination Policy
      with Additional Effects Included ......................................................................... 252
List of Tables

3-1 Basis for Important Model Structure Components ........................................ 115-116
3-2 Single Phase Sensitivity Test Parameter Values ........................................... 124
3-3 Single Phase Sensitivity Test Results
Project Performance Under Pessimistic and Optimistic Scenarios .................... 125
3-4 Single Phase Sensitivity Test Results
Project Performance in Percent Change From Likely Scenario
Under Pessimistic and Optimistic Scenarios ..................................................... 126
3-5 Single Phase Sensitivity Test Results
Total Sensitivity to Parameter Change ............................................................. 128
3-6 Multiple Phase Sensitivity Test Parameter Values ......................................... 130
3-7 Multiple Phase Sensitivity Test Results
Project Performance Under Pessimistic and Optimistic Scenarios .................... 131
3-8 Multiple Phase Sensitivity Test Results
Project Performance in Percent Change From Likely Scenario
Under Pessimistic and Optimistic Scenarios ..................................................... 132
3-9 Multiple Phase Sensitivity Test Results
Total Sensitivity to Parameter Change ............................................................. 133

5-1 Python Project Interfaces of Development Phases and
Functional Groups ......................................................................................... 171
5-2 Project Variable Types Generated by Interviews ........................................... 177
5-3 System Description Parameter Values
for Multiple Phase Model Calibration ............................................................ 197
List of Tables

5-4 Model Parameter Values for Three Coordination Policies ......................... 237
5-5 Project Performance Using Three Coordination Policies .......................... 240

6-1 Management Focus in Traditional and Expanded Project Models .............. 262
Chapter 1

Introduction

1.1 Context

The Bose Corporation needed improved product development performance. Increased competition and customer requirements called for the faster development of better integrated sound system products. So Bose re-engineered the early portion of its product development process for integrated sound systems. Before re-engineering Bose used a functional organizational structure which separated marketing, engineering, and manufacturing into three departments. These departments passed deliverables from marketing to engineering and from engineering to manufacturing with little interdepartmental coordination. Bose applied the Concept Engineering (Burchill, 1992) approach to re-engineer its process and organization to improve quality and reduce cycle time in the face of increased product complexity. Marketing and concept design functions were combined into a new Concept Development group. Conceptual designs and business plans are passed from Concept Development to the new Detailed Design group. Final designs are passed from the Detailed Design group to manufacturing. The flow of products through Bose's product development process was also re-engineered with new
deliverables and milestones for the new development groups. Bose expected significantly higher quality products and reduced cycle times from the re-engineering.

However re-engineering the process and organization doesn't completely solve development project performance problems. Re-engineering at many companies appears to have altered the appearance but not the effective content of their product development efforts. Often only one or two performance measures improve (Cooper and Kleinschmidt, 1994). Sometimes performance deteriorates. For example, re-engineered development systems have failed to reduce (Nevens et al., 1991) and have on occasion increased cycle times (Burchill and Walden, 1994; Clark and Fujimoto, 1991a). Bose experienced improvement in the integration of marketing and conceptual design functions by re-engineering. But reductions in cycle time did not materialize. Coordination problems between the Concept Design and Detailed Design groups replaced similar problems between marketing and engineering. For example concept designs were considered unfeasible by the Detailed Design group. Ineffective communication between product developers and management also contributed to coordination problems. Frequent changes in project objectives and available resources reduced development efficiency and increased rework. Re-engineering the product development process and organization had moved these issues from the marketing-engineering interface to the Concept Design-Detailed Design interface. Bose's re-engineering of their process structure may have primarily shifted the location of their coordination problems instead of solving them. These problems prevented Bose from experiencing the hoped-for reductions in cycle times.

What prevented this industry leader renowned for quality and research from gaining the full and expected benefits from a thorough re-engineering effort? What product development components, characteristics and relationships generated counterproductive behavior in well-intentioned product developers? How can the product development process be described and investigated to better understand these issues?
Bringing new products to market which fulfill customer needs is critical to success in open competitive markets (Patterson, 1993). Development projects generate these new products for market introduction. Developing products faster, of higher quality, and cheaper than competitors can increase market share, profit, and long term competitive advantage. This has made the performance of product development projects an increasingly important area of competitive advantage in many industries. For many years development schedules, quality, and cost have been managed primarily with traditional project management tools such as the critical path method. However changes in competitive conditions within the last decade have increased the difficulty of accelerating projects, increasing quality, and reducing costs with traditional project management tools.

Three recent changes in competitive conditions have increased the imperative to and difficulty of improving project performance (Wheelwright and Clark, 1992; Patterson, 1993). First, competition has grown increasingly international, diluting or eliminating geographic protection for many firms. This has forced firms to compete with the best in the world instead of the best in their market area. Globalization of firms has combined with increased access to international markets to make competition more intense, more demanding, and less forgiving of errors. Second, customers have grown more sophisticated and demanding of products. Standards for performance, ease-of-use, and reliability are high and rising. Customers demand increased variety and thereby fragment previously homogenous markets. Third, the pool of technologies used to develop new products is growing. This has increased the number of possible solutions to customer needs and can transform the development process itself.

Many manufacturing firms have adopted a new product development paradigm and re-engineered their product development processes and organizations in response to the challenges of product development (Rosenthal, 1992). New systems such as concurrent engineering and cross-functional development teams have replaced more sequential and functional-based systems to integrate product development efforts and improve
performance. Likewise, construction firms are using new technologies such as computer assisted design to accelerate and integrate the development process. These new conditions and approaches have impacted product development in at least three ways. Product development environments have become increasingly dynamic (Wheelwright and Clark, 1992; Nevens et al., 1991). Concurrent development (Clark and Fujimoto, 1991b) and increased awareness of the influence of dynamic relationships on successful project development (Wetherbe, 1995; Iansiti and Clark, 1993; Osborne, 1993; Patterson, 1993; Rechtin, 1991) have increased the role of dynamic effects on project performance. Finally, the number of interdependencies among development activities and participants has increased (Hayes et al., 1988). This has led both researchers and practitioners to recognize the increasingly critical role of understanding the dynamics of projects in managing the development process for success.

While some firms have attributed significant improvements to the adoption of the new paradigm and methods (Merrills, 1991; Nevins and Whitney, 1989) the aggregate results have been mixed (Iansiti, 1993a; Clark and Wheelwright, 1993; Dean and Susman, 1991). One contributing factor is that these systems require more coordination than traditional processes and organizational structures to develop affordable high quality products quickly (Iansiti, 1993b; Clark and Wheelwright, 1993). This may be because the new paradigm has increased the complexity of the product development process as described above and thereby tightened the constraints imposed by the interdependencies among participants and processes (Ulrich and Eppinger, 1994; Malone and Crowston, 1990). These increased complexities become apparent at the project level where project managers attempt to operationalize departmental designs and policies to develop specific products.

This research investigates the dynamic impacts of product development process structure and project coordination policies on performance. Although model validation will focus on the development process of a specific manufacturing industry (semiconductors) the
project structures are common to many development projects. The motivation for the research, problem definition, and model development draw on several industry processes. The applicability of the research framework and results across industries is discussed in the conclusions.

1.2 Motivation for Research

Competitive forces such as intense global competition, fragmented and demanding markets, and diverse and rapidly changing technologies cause companies to view improved product development as a competitive imperative (Wheelwright and Clark, 1992). These forces have increased the complexity and uncertainty of product development. The product development processes and organizations created for relatively stable markets, long product life cycles and project durations, and technology-based competition are often no longer capable of producing products fast enough, inexpensive enough, and of high enough quality to remain competitive (Clark and Fujimoto, 1991a). Entire industries are re-engineering how they develop products (Irving, 1993; Peterson and Hillkirk, 1991).

Therefore improving project performance may not be as simple as it first appears. Well-intentioned changes to the development process can cause severe unintended side effects (Thomas and Napolitan, 1994; Jessen, 1992). An example using the increase in headcount to improve schedule performance follows. These effects can be amplified by time delays and nonlinear relationships among project components (Cooper, 1993b). Part of the cause of the difficulty lies in the internal structure of the development process (Richardson and Pugh, 1981; Roberts, 1974) and the coordination policies used to manage the product development activities and resources (Hoedemaker et al., 1994; Iansiti and Clark, 1993; Fujimoto et al., 1992). Companies which experience difficulty coordinating their development efforts also have long cycle times (Clark and Fujimoto, 1991a). Those which coordinate well also have short cycle times (Takeuchi and Nonaka, 1991; Clark
and Fujimoto, 1991a). Iansiti and Clark (1993) found that more internal integration (coordination) was positively related to performance as measured by quality, productivity, and lead time in the development of automobiles. As a specific example, Bose experienced difficulties in coordinating the work of the Concept Design and Detailed Design groups (Ford et al., 1993). The Concept Design group needed assistance from the Detailed Design group to meet the review and approval standards set by management for Concept Design deliverables. Concept Design found it difficult to obtain this assistance from Detailed Design before the hand-off because management had not approved the Concept Design deliverables and therefore had not authorized Detailed Design to work on the project. This created a circular problem for the Concept Design group: it needed Detailed Design help to get approval and needed approval to authorize Detailed Design to help. Detailed Design also experienced coordination difficulties with the re-engineered process and organization. Conceptual product changes continued after Concept Design had given the project to Detailed Design. These changes required unanticipated coordination by both groups and rework of designs by Detailed Design. Another problem was that the priorities of Concept Design, Detailed Design, and management were not aligned. In one case the misalignment of goals resulted in a severe lack of shared ownership in two projects by different groups of developers. This generated project sabotage and the failure of both products to pass Concept Design reviews and receive approval to continue development. The misalignment of priorities also forced Detailed Design personnel to regularly move from one project to another in attempts to coordinate their priorities with those of management. Problems in the coordination of development resources can prevent improvements to development project performance which were intended by changes to the process structure.
1.3 The Problem

Static features and impacts of projects have been extensively researched and applied to project management practice (e.g. Barrie and Paulson, 1984; Moder et al., 1983; Halpin and Woodhead, 1980). In contrast, project managers do not effectively understand or utilize the dynamic features of development project structures (Cooper, 1994, 1993a,b,c, 1980; Cooper and Mullen, 1993; Sterman, 1992; Reichelt, 1990; Brooks, 1978). These dynamic features include feedback systems, time delays, and nonlinear cause-effect relationships among project components. These features combine to cause project systems to behave in complex ways which are difficult to understand, predict, and manage.

A simple example demonstrates the potential effects of feedback, time delays, and nonlinear relationships in project structures. Consider a project in which the expected completion date exceeds the deadline, creating a schedule gap. A common managerial response is to increase headcount (number of designers or crews) to increase output, move up the completion date and thereby reduce the schedule gap. This simple feedback structure can be described with the causal loop diagram (Goodman, 1988, Richardson and Pugh, 1981) shown in Figure 1-1. In causal loop diagrams casual links (arrows) are labeled as those which cause the variable at the arrowhead to move in the same (+) or opposite (-) direction as the variable at the arrow's tail, when other factors are held constant. Feedback loops are labeled as balancing (B) if variable values tend to be goal-seeking over repeated passes around the loop or reinforcing (R) if repeated passes accelerate movement in a single direction (Richardson, 1986; Richardson and Pugh, 1981).1

1 A more rigorous definition of causal link and causal loop polarity is available in Richardson (1995).
Figure 1-1: A Development Project Feedback Structure

The feedback structure in Figure 1-1 describes how the project condition (the size of the schedule gap) influences the managerial response to the system (change in headcount), which in turn affects the condition of the system (reduced schedule gap). In isolation the feedback structure in Figure 1-1 would restrain the project's schedule gap. But the feedback inherent in complex systems such as development projects often has many unintended side effects. For example Thomas and Napolitan (1994) identified fourteen secondary impacts of changes in construction development projects caused by three primary impacts (increased costs, schedule delays, and rework). Those fourteen secondary impacts are:

- Decreased worker productivity
- Lowered design team morale and productivity
- Relocation of labor
- Increased planning, coordination, and rescheduling activities
- Possible out-of-sequence work
- Demobilization, remobilization
- Overtime (fatigue) due to acceleration
- Crowding due to acceleration
- Possible seasonal/weather related impacts due to delays
- Increased effort to price out and negotiate the changes
- Learning curve associated with a change
- Inadequate coordination of changes
- Additional value engineering due to increased costs
- Possible litigation

An unintended side effect of increased time required to coordinate larger headcounts can be described with the reinforcing causal loop shown in Figure 1-2.

![Causal Loop Diagram](image)

**Figure 1-2: An Unintended Side Effect of a Project Management Policy**

The unintended side effect shown with the reinforcing loop in Figure 1-2 counteracts the intended impact of the balancing loop. This is because some of the increased headcount is used to address the increased coordination need instead of increasing output. If the unintended side effect is larger than the intended effect it can extend the Expected Completion Date, increasing the Schedule Gap. This could occur immediately after implementation of the increased headcount policy. The relative strength of the balancing
and reinforcing loops at any given time determines whether the Schedule Gap is increasing or decreasing. Which feedback loop dominates the system behavior is strongly influenced by another characteristic of dynamic systems, time delays. For example, a delay in the direct influence of Headcount on Expected Completion Date can cause the reinforcing loop to dominate soon after the headcount increase and the balancing loop to dominate later. Shifts of dominance among the feedback loops in a project structure cause project behavior to oscillate and can magnify impacts (Diehl and Sterman, 1995; Richardson, 1995; Forrester, 1961).

Nonlinear relationships among components is a third important characteristic of dynamic systems. An exponential relationship between Headcount and Percent of Time Required for Coordination is shown in Figure 1-3.

![Figure 1-3: A Delay and A Nonlinear Relationship in a Project System](image-url)
Nonlinear relationships make systems difficult to predict and manage by causing the system to respond differently to the same managerial action depending upon the system's current condition. For example, an increase in Headcount by 10% would generate a very small increase in Percent of Time Required for Coordination if the Headcount was small (left side of the Headcount versus Percent Time curve). But the same 10% increase in Headcount would generate a large increase in the Time Required for Coordination if the Headcount was high (right side of the Headcount versus Percent Time curve).

When project structures are described with causal loop diagrams management policies can be viewed as plans which attempt to alter the strength of causal link relationships between variables or create or delete feedback mechanisms represented by loops. In this way management policies can influence the relative dominance of different feedback loops. For example, a project manager may quickly add more people to a project when it gets behind schedule to increase the influence of the balancing loop. If the manager recognizes the potential loss of net productivity due to more time being needed to coordinate the work new technology which facilitates coordination may be introduced. This new technology is intended to weaken the relationship depicted by the arrow between Headcount and Percent Time Required for Coordination and thereby weaken the reinforcing loop which slows progress.

The combination of feedback, time delays, and nonlinear relationships in project structures have been shown to reduce performance and cause them to be very difficult to manage in the construction industry (Thomas and Napolitan, 1994; Reichelt, 1990; Cooper, 1981). The dynamic nature of project behavior precludes the generation of a single set of decision rules which are robust in the face of all possible project conditions. Project managers must use their understanding of project systems to adjust management policies such as those for coordination to specific project circumstances and the evolution of project behavior. This requires that development managers include dynamic features in their project mental models. But the mental models used to describe, explain, and predict
projects do not generally include the dynamic features. Both complexity and dynamic features of projects are poorly understood by managers (Diehl and Sterman, 1995; Sterman, 1994; Paich and Sterman, 1993; Rechtin, 1991). The resulting inadequate project mental models prevent the development of decision heuristics which incorporate dynamic features into project management decisions. This deficit in decision heuristics therefore constrains project performance.

The underlying problem addressed by this research is the failure of project managers to fully recognize and utilize the dynamic features of projects which often drive project performance. Managers cannot effectively manage projects without understanding the impacts of dynamic features. The understanding and use of project dynamics which are currently used remains trapped in the intuition of experienced managers. An improved understanding of project dynamics is a first step in improving project mental models, decision heuristics, and thereby project performance. This research seeks to improve that understanding by increasing our knowledge of how product development process and coordination policy impact project performance. Developing a tool for an improved understanding of these impacts is the focus of this work. Therefore the research question is "How does development project structure impact project performance?" This question will be investigated through the building and validation of a dynamic simulation model of a development project and the use of that model to investigate a coordination policy for improved project performance.

1.4 Research Approach

The purpose of this research is to increase the knowledge and understanding of development projects. This improved understanding can act as the basis for improved project mental models, management heuristics and decisions, and project performance. While no single approach or model can provide a complete understanding of development projects, this thesis will contribute by identifying feedback relationships and
other dynamic features which significantly impact project performance and by evaluating the nature of those impacts. This will be done by integrating existing but previously separate project structures into a single model and by expanding and testing previously unavailable project structures which have potentially large impacts on performance. The model is then applied to the investigation of a significant project management issue, coordination.

This thesis uses dynamic computer simulation to model and investigate the impact of development processes and coordination policies on project performance. A computer simulation model provides several advantages. First, the many and various project parameters and relationships can be modeled more comprehensively with the flexible representation available than with manual modeling methods. Second, assumptions are made explicit and unambiguous by their representation as formal equations. Third, consequences of assumptions and policies over time can be revealed through the simulation under safe experimental conditions. Finally, the model's reflection of actual project structures provides an effective means of communicating research work and results.

This research focuses on the development of a specific group of products, tangible durable products which evolve through a series of steps through the efforts of several specialists. This group includes many development processes in many industries. A generic set of activity names for the process being studied could include: 1) identification of need, 2) conceptualization, 3) product definition, 4) design, 5) testing, and 6) ramp-up to operations. The names and levels of detail which describe those steps vary widely among industries. For example one semiconductor development process uses the terms Market Study, Product Definition, Design, Pilot Product Solution (prototype testing), and Pilot Product Testing (manufacturing process testing). The relative importance of the different activities within the entire process can also vary widely. For example in the semiconductor industry ramping up to steady state production can take only a few weeks
out of years of total development and is relatively inexpensive. In contrast the same activity in the real estate and construction industries is called construction and often takes as long as all other activities combined and costs many multiples of all other project costs. Despite these differences the model described herein can reflect the processes of many industries including manufacturing, real estate and construction, software, book publishing and feature film making.

The system dynamics methodology (Forrester, 1961) for modeling complex systems has been adopted. System dynamics describes cause and effect relationships with stocks, flows, and feedback loops. Stocks and flows are used to model the flow of work and resources through a project. Information feedback loops are used to model decisions and project management policies. Actual, desired and perceived conditions are explicitly and separately modeled. Time delays such as between the need for a development activity and the availability of labor to perform the activity are explicitly identified, as are nonlinear relationships. The methodology provides the means of describing the dynamic structures of development projects and therefore is an excellent foundation for this research.

1.5 Summary

The successful performance of product development projects is critical to competitiveness in many industries. Recent market and technology changes have increased the importance and difficulty of improving project performance. Although understanding the impacts of the dynamic aspects of development projects is increasingly important for improvement, these features are typically unrecognized, ignored or used inappropriately. An improved understanding of dynamic project features is needed to improve project mental models, decision heuristics, and thereby performance. This research contributes to this understanding by building and validating a system dynamics model of a development project and applying it to the investigation of coordination policies.
Chapter 2

Literature Review

2.1 Introduction

This chapter describes and evaluates the literature as it pertains to this research. Traditional characterizations of product development projects are described, followed by characterizations which have appeared over the last decade. These descriptions provide the basis for the survey and evaluation of product development models for investigating the dynamic impacts of process structure on performance. This is followed by a more detailed evaluation of existing system dynamics models of projects for their contribution to this work. Finally, gaps in the current literature are identified as the starting point for the specific work of this research.

2.2 Traditional Characterizations of the Product Development Process

The traditional model of the product development process and organization is based upon a sequential and functional approach to development (Wheelwright and Clark, 1992;
Zaccai, 1991). In the traditional paradigm the development process is a series of development activities from conceptualization to product introduction. An example of the traditional development process can be seen in Boeing's description of their development process for the 727-100 airliner in the 1960s (Maxam, 1978):

"There were three distinct formal Engineering phases, following an informal series of concept studies, of the successful airplane program. Although these phases appeared to overlap and blend together this appearance was caused more by the individual scheduling of each element of the design than by the actual blending of the phases."

Boeing's description reveals their intention to keep the development phases separate and sequential. Many researchers have described the traditional process and given examples from industry (e.g., Wheelwright and Clark, 1992; Womack et al., 1990; Nevins and Whitney, 1989; Hayes et al., 1988). Clark and Fujimoto (1991b) describe this paradigm as appropriate "...when markets were relatively stable, product life cycles were long, and customers concerned most with technical performance."

Substandard project performance under the traditional paradigm generates friction among functional groups, little and poor coordination, and bottlenecks in the flow of products through the development process (Ulrich and Eppinger, 1994; Hayes et al.; 1988). This can extend cycle times or incur additional resource use, thereby increasing costs. Increased resource use can be seen in Boeing's description of how its 727-100 program responded to changes: "...even after the Design Development Phase was underway for some time, the designer was still faced with many changes as more and more technical data was being generated, usually as the result of the testing process. It is also interesting to note that the Engineering schedules did not have the luxury of spare time, thus these late breaking changes had to be accommodated through overtime, work around, and a high dose of ingenuity of all concerned both inside and outside the Engineering organization." (Maxam, 1978).
In the traditional product development paradigm the three traditional measures of project success - time, cost, and quality/scope - are increased or decreased to improve total project performance. This can take the form of trading performance among the three measures in a zero-sum environment (Rosenau and Moran, 1993). But more mutually beneficial changes are often also available. This approach can be seen in Boeing's reduction of scope in the development of the 777 aircraft. Boeing completely avoided full-scale physical mock-ups of its 777 airplane by designing and testing with software (Stix, 1991). Gomory (1991) provides another example of reducing cycle time using scope reduction. A team of developers at IBM used previously developed standard components in the development of computer terminals instead of developing new components, thereby reducing cycle time by five months.

The Critical Path Method and PERT (program evaluation and review technique) are two traditional tools which are widely used to manage development projects. Although initially developed for schedule control they have been expanded to manage resources (and therefore costs). They are based upon the traditional paradigm of development. The Critical Path Method disaggregates the development process into activities which are related through their temporal dependencies. Each activity is treated as a monolithic block of work described only by its duration. The temporal dependencies describe the constraints which earlier (upstream) activities impose on later (downstream) activities. The constraints are described with relationships between the beginnings and completions of activities. The logic of the schedule can be represented in a network diagram. A simple example of a network diagram is shown in Figure 2-1.
Critical Path Method calculations identify a project's critical path, which is the sequence of tasks whose combined durations define the minimum possible completion time for the entire set of tasks (Ulrich and Eppinger, 1994). Earliest and latest possible start and finish dates of all activities within a schedule determined by the critical path can be calculated, as can the available slack times. The results of this planning and analysis can be presented for broader communication with a Gantt chart. An example of a Gantt chart from Moder et al. (1983) is shown in Figure 2-2.
The Critical Path Method provides several tools for trading away good performance in one measure for improved performance in another. For example, durations of activities along the critical path can be shortened by adding more resources (Ulrich and Eppinger, 1994; Wheelwright and Clark, 1992; Moder et al., 1983). The Critical Path method provides a time-cost-tradeoff method for analyzing the effectiveness of accelerating alternative activities. The effects of altering activity dependencies among activities to shorten the critical path can be investigated (Barrie and Paulson, 1984; Moder et al., 1983).

The Critical Path Method is easily understood and applied. It provides a set of fundamental tools for characterizing and managing a development project in temporal terms. However, the method has critical limitations. The method assumes no rework of errors which are undiscovered when the phase is "completed." and that the rework of errors discovered within a phase's duration is incorporated into the phase duration estimate. The method cannot explicitly represent bilaterally coupled activities and therefore cannot describe loops, feedback, or iteration in a system. It also assumes that the development project remains unchanged over time. This prevents the method from modeling time-varying and endogenous factors such as developer skill, training, and coordination issues. Therefore, the Critical Path Method is unable to model the highly coupled aspects and dynamic nature of the product development process. Finally, the Critical Path Method cannot describe the rational which underlies the structure description and therefore lacks depth of information content.

PERT (Project Evaluation and Review Technique) uses an approach to schedule management which is similar to the Critical Path Method. This method was developed for processes such as product development (Moder et al., 1983). PERT addresses one of the limitations of the Critical Path method by incorporating the uncertainty inherent in the estimates of the durations of development activities into a scheduling tool. Three estimates of project duration are used for each activity to model the variability of
durations. The PERT method calculates the probabilities of a project meeting specific schedule objectives. PERT incorporation of duration uncertainty makes it more valuable in managing less certain processes such as product development. However PERT requires lots of data and is limited in accuracy by the estimates of variability of activity durations. Like the Critical Path method, PERT cannot explicitly represent coupled loops or feedback, assumes the project is static, and cannot model causes of process behavior.

2.3 Recent Characterizations of the Product Development Process

Market and technology changes described previously and the limitations of traditional project management methods led to the development of a new image of effective product development (Nevins and Whitney, 1989; Hayes et al., 1988). Although this new image is still emerging its central features have been articulated by researchers and applied by industry. The new paradigm fundamentally alters both the product development process and organization. Researchers currently envision product development as a collection of highly coupled activities which are performed iteratively and often simultaneously by cross functional product development teams (Ulrich and Eppinger, 1994; Wheelwright and Clark, 1992; Womack et al., 1990). The dominant change in the development process from the traditional to the new paradigm is from sequential activities to concurrent activities (concurrent development). The dominant change in the development organization from the traditional to the new paradigm is from functional departments to cross-functional development teams.

2.3.1 Concurrent Development

Concurrent development's primary purpose is cycle time reduction. Concurrent development improves cycle times by planning, facilitating, and executing multiple development tasks simultaneously instead of sequentially as in the traditional development paradigm. This requires breaking each development activity into more
smaller tasks and starting downstream tasks as soon as all prerequisites are available. Figure 2-3 illustrates the fundamental difference between the traditional ("Phased Approach") development process and concurrent design ("Overlapping Approach").

![Diagram of Phased and Overlapping Approaches]

**Figure 2-3: Sequential and Concurrent Product Development Processes**  
*(Hayes et al., 1988)*

Boeing moved from the traditional development process used to develop the 727-100 to a concurrent approach for the development of its 777 aircraft. To apply concurrent development in the design portion of the project Boeing overlapped its fifteen previously-sequential design and engineering steps to allow work to proceed on many development activities simultaneously (Stix, 1991). The concurrent approach was also applied at the project level. For example Boeing separated the product definition phase into three tasks and began major assembly of the aircraft before the product definition was 90% complete (Peterson and Sutcliffe, 1992).

Large reductions in cycle time can be realized by applying concurrent development (Wheelwright and Clark, 1992, Womack et al., 1990; Nevins and Whitney, 1989). But the
cycle time reduction comes at the cost of increased complexity. The disaggregation of
development activities into smaller tasks increases the number of dependencies and the
number of required information transfers. Because downstream tasks are often started on
incomplete information the number of design iterations (rework) is also increased.
Boeing's development of the 777 can illustrate the impacts of concurrent design. Boeing's
process description for the 777 explicitly includes eleven iterative loops among nine
development phases (Peterson and Sutcliffe, 1992). The additional coordination required
for Boeing to use concurrent development was enormous. Boeing considered traditional
development information systems wholly inadequate and therefore developed the 777
completely on computers. Coordination of the work of 4,000 engineers on the 130,000
different parts of the airliner is a major reason (Stix, 1991). As illustrated by the Boeing
777, managing increased iteration is an important issue raised by the use of concurrent
development.

Steward (1981) and Eppinger et al. (1990) developed the Design Structure Matrix to
investigate the iterative nature of product development. The Design Structure Matrix is a
square matrix with the full set of development activities as both row and column labels.
Each cell within the matrix represents a unidirectional dependency between two
activities. Design Structure Matrices have been used to map (Smith and Eppinger, 1991)
and predict (Morelli and Eppinger, 1993) information flows among activities. The matrix
can be used to identify information flows as sequential, parallel, or coupled and for the
efficient ordering of development activities. Chao (1993) applied the Design Structure
Matrix to study the use of iterations in making time/quality tradeoff decisions. The focus
was a portion of product development at a large semiconductor firm (DEC). Two
strategies for making time/quality decisions (faster iteration and higher quality iteration)
were proposed and tested.

Osborne (1993) applied iteration maps and the Design Structure Matrix to describe
product development at a leading semiconductor firm (Intel) in terms of cycle time.
Osborne investigated variability in cycle times. His conclusions about the impacts of iteration on cycle time are pertinent to this research. They include:

"Iteration is a significant component of product development cycle time, typically about one third of project effort, but can represent as much as two thirds or as little as 13% of project effort.

Few variables independently influence cycle time. Major project iterations are significant. Another key variable correlating with cycle time is project complexity in terms of man hours necessary to develop the product.

...Iterative modeling tools provide a means to think about the impact of changes on the total system."

Osborne's work demonstrates the need for additional investigation of the impacts of dependencies among development activities on cycle time. It also points to the need for a better understanding of how variables which impact cycle time can be identified and managed (coordinated). The Design Structure Matrix is a potentially useful tool in describing and investigating information transfer and iteration for cycle time reduction. But the Design Structure Matrix cannot directly model the structure of a development process over time. Like the Critical Path method, the Design Structure Matrix assumes that the dependencies between phases are fixed or that the distribution is fixed. Osborne's research supports other work which suggests that iteration in product development is a primary cause of the dynamic nature of product development process (Cooper, 1994, 1993a,b,c; Ford et al., 1993; Seville and Kim, 1993; Kim, 1988). Iteration is therefore suspected to be a primary driver of cycle time performance as well as a measure of quality.

2.3.2 Cross-Functional Development Teams

A primary purpose of cross-functional teams is improved quality and effectiveness through improved coordination. Cross-functional development teams are groups of development specialists from different functional domains who work together on a single
development project. The formation of cross-functional development teams is an extension of the move away from functional-based groups to the matrix structures used in the traditional development paradigm. Hayes et al. (1988) describe and Wheelwright and Clark (1992) later refine a more detailed model of this shift with intermediate steps defined by the level of influence of project managers. Restructuring product development organizations away from function-based groups and toward cross functional development teams has also become a widely used approach to reducing cycle time (Clark and Fujimoto, 1991b).

Boeing's 777 project provides an example of cross-functional development teams (Peterson, 1992; Stix, 1991). Boeing modified its matrix structure for the 777 project. Chief Engineers lead functional domains such as propulsion, avionics, structures, electrical, flight deck, and aerodynamics. They are responsible for functionality, reliability, maintainability, manufacturability, cost, and certification. Chief Project Engineers are responsible for the integration of at least one of the airplane's sixty-five individual systems. Additional Chief Project Engineers integrate these individual systems within the airplane as a whole and integrate the development project with external participants such as customers and certification testing organizations. Boeing formed over 270 cross-functional development teams within this structure. Peterson's (1992) description of the teams illustrates the cross-functional nature of their role: "These teams are defined around individual airplane systems, and are working cross-systems integration and vertical development issues (life cycle) simultaneously."

However several researchers (Bacon et al., 1994; Clark and Fujimoto, 1991b; Dean and Susman, 1991; Takeuchi and Nonaka, 1991) and many firms (e.g., see Clark and Fujimoto, 1991b, pg. 105) have realized that the formation of cross-functional teams alone does not improve cycle time. Wheelwright and Clark (1992) cite a case in which unsuccessful cross-functional teams increased cycle times. They identify overextended managers as a contributing factor in cross-functional team failures. Reasons cited by
other researchers vary. Dean and Susman (1991) found friction between members of the team from design and manufacturing. Wheelwright and Sasser (1991) cite a lack of planning due to a lack of information. Nevens et al. (1991) identified a lack of cross-functional skills in team members and no one taking responsibility for coordination. Clark and Fujimoto (1991b) found an automobile development team consisting of only liaison people and no developers. The team failed because it was ignored by those developing the product.

The new development paradigm addresses the increased coordination needs of projects with cross-functional development teams. The apparent assumption is that cross-functional teams address the dynamic drivers of cycle time better than traditional structures. But the mixed results of applying cross-functional teams for improved project performance indicates that the use of cross-functional development teams does not adequately address dynamic aspects of development projects.

2.4 Summary of the Product Development Process Literature Review

The existing literature describes and documents recent fundamental changes in product development processes and organizations from a sequential functional approach to a concurrent team approach. In doing so it tightly links dynamic project features such as iteration with project performance. But the existing research does not described in detail or explain the relationship between dynamic features and performance. This deficit is apparent in the industry in which product development has been studied most extensively, the automobile industry. Cusumano and Nobeoka (1991) identify the need for additional research concerning coordination and cycle time in their review and critique of research of product development in the automobile industry. The current research develops an improved understanding of the relationship between dynamic project features and performance and how coordination can be used to improve performance.
2.5 System Dynamics Literature Review

Several models using the system dynamics methodology (Forrester, 1961) incorporate dynamic features into models of single product development projects. The feedback structures of system dynamics models describe the modeler's hypotheses about the dynamic behavior of the project and form a framework for describing their investigations of project behavior. The key loops abstracted from these models have been aggregated into six feedback structures.

2.5.1 Six Key Feedback Structures in System Dynamics Models of Projects

2.5.1.1 The Labor Structure

The labor structure includes three balancing feedback loops which increase labor effort as a response to schedule pressure is common in project management and central to several models of product development. The response can adjust the amount of effort (Labor Quantity), how hard people work (Labor Intensity) or both as shown in Figure 2-4.

![Figure 2-4: The Labor Structure](image-url)
Product development models have explored these fundamental relationships more deeply in several directions. Some of the variables and relationships describe intermediate variables in the linkage between management policies and project performance. Projections of labor needs have been modeled by Cooper (1980). The impacts of Labor Quantity on Labor Intensity have been studied by Reichelt (1990). Cooper (1980) added types of labor and characteristics of labor. Abdel-Hamid (1984) used the impact of experience on labor productivity. Jessen (1990) investigated project manager motivations with a resource-based model. Development resources other than labor (e.g. construction materials or testing machines) can be modeled with the same structure as the Labor Loops (e.g. Cooper, 1980). However existing models generally assume that labor is the dominant resource in product development projects.

The Labor Structure provides a method of describing the impacts of certain management policies. Those policies can be viewed as plans for altering the strength of certain relationships between variables which are represented by the arrows. For example, a project manager may quickly add more people to a project when it gets behind schedule. This policy is attempting to increase the strength of the causal link between the variables "Schedule Pressure" and "Headcount" and therefore increase the influence of the middle feedback loop.

2.5.1.2 The Schedule Structure

The schedule structure describes another common project management tool, slipping the deadline in response to Schedule Pressure (Figure 2-5). Many models include this feedback loop (Roberts, 1974; Richardson and Pugh, 1981; Abdel-Hamid, 1991).
Figure 2-5: The Schedule Structure

The Schedule Structure uses a floating goal structure to describe a common feature of project management practice. In a floating goal structure the goal (the Deadline Date) drifts toward the system conditions (the Expected Time to Completion). In this example the purpose is to reduce the Schedule Pressure. This drift is slowed by the Resistance to Slip Deadline Date, which can represent the development team's commitment to schedule performance. Project management policies can change the influence of the schedule structure on project performance by altering the Resistance to Slip Deadline Date, for example with liquidated damages which impose a cost on the project for slipping the deadline date.

2.5.1.3 The Rework Structure

Rework is the correction of errors required to make the product functional. It is distinguished from quality, which has its own loops, by the fact that rework must be done, whereas quality work is optional. For example, work to fix a software bug which
prevents the saving of a word processing document is rework, whereas work to accelerate the saving of a word processing document augments quality. Rework is a part of all large development projects, although the amount varies widely (Cooper, 1993a,b,c). The Rework Structure describes the effects of this additional effort on a project's progress toward completion (Figure 2-6).

![Figure 2-6: The Rework Structure](image)

The balancing loop in Figure 2-6 represents the intended impact of a management response to an increase in schedule pressure - reduce the work remaining. The two reinforcing loops represent the impacts of the unintended side effects of the rework structure - the generation of additional errors which require correction.

Simpler versions of the Rework Structure have driven some project models (Seville and Kim, 1993; Jessen, 1990; Kim, 1988). But the separate modeling of Undiscovered Errors by Cooper (1994, 1993a,b,c, 1980) and others has added significant new insights into the impacts of policies addressing rework on project performance. Cooper (1993a) estimates
the delay in the discovery of errors to be approximately 1/4 to 3/4 of the time required to
design the work the original time and concludes that this delay is one of the most
important determinants of cycle time performance. Management policies can have a large
impact on performance by reducing this delay, effectively weakening the influence of the
lower reinforcing loop. Those policies could be rudimentary forms of communication
between project phases such as quality assurance checking for errors or development
team meetings. Coordination policies which increase flexibility may also weaken this
loop though these relationships.

2.5.1.4 The Available Work Structure

The performance of projects can be constrained by the availability of work. Some phases
of projects have internal constraints on work availability. For example, in the steel
erection phase of a high-rise building project the second floor steel work cannot be done
until the first floor steel work is completed. In multiple-phase projects performance can
also be constrained by the amount of work released by preceding phases. For example,
first floor steel erection cannot begin until the first floor steel is fabricated, which cannot
begin until that steel is designed.

The limited availability of work to an individual phase of a project is the basis of some
models of projects (queuing theory and the critical path and PERT methods). The
constraints imposed by the available work impact the rate of basework completion
through the availability of incomplete basework as shown in Figure 2-7.
Figure 2-7: The Available Work Structure

Basework is work being performed on development tasks for the first time. The reinforcing loop in the available work structure represents the release of new work for completion due to completing work within the development phase or by an upstream development phase. The balancing loop in the available work structure represents the reduction in the basework available but not completed (the work queue) due to the completion of work. The Upstream Tasks Released and Precedence Constraints are the basis for modeling multiple interdependent phases of a development process. These precedence relationships describe the progress possible by a downstream phase based upon the progress made by the upstream phases on which it is dependent (external constraints) and the progress possible based upon progress within the phase itself (the phase's internal constraint). The critical path and PERT models are based on descriptions of these external constraints which relate the start and finish times of phases (Moder et al., 1983). Typical relationships are independent (no relationship), sequential, and concurrent with a difference in start times. Cooper (1980) describes a large system dynamics model of shipbuilding in which each development project has at least seven phases. In this model Availability of Prior Work is explicitly modeled as a constraint on Subsequent Work Progress although the specific structure is not available. Reichelt
(1990) expands the description of this relationship between the engineering and construction phases and relates it to the effects of design changes. Richardson (1982) and Richardson and Paich (1981, 1980) used product development as the first of two phases in the investigation of cycle time. These models were built to replicate specific processes and are therefore very project-specific. Homer et al. (1993) subsequently developed more general descriptions of the constraints on work progress imposed by both preceding phases and the work within the phases itself, as will be described in the next section. Two models (Seville and Kim, 1993 and Ford et al., 1993) have attempted to generalize the phases of a development project. In both cases only two phases were modeled and the relationships between the phases were not generalized.

2.5.1.5 The Quality Structure

The quality structure represents the project management functions which effect the optional repetition of development tasks (Figure 2-8). The optional or flexible nature of the work reflected in the quality structure distinguishes it from the rework structure, which is required for basic product functionality. The Quality Standard reflects the level of fulfillment of customer objectives (Fiddaman, Oliva and Aranda, 1993) as well as the number of imperfections released with the product. The quality structure reflects a very real aspect of project management which is not reflected in the rework structure: the voluntary setting of a product performance standard and adjustment of the development process to meet that standard. This can be particularly influential because project managers can influence voluntary iteration with their decisions but must perform rework.
Figure 2-8: The Quality Structure

The right reinforcing loop in the quality structure represents one of the intended impacts of setting quality standards - improve product quality and raise the quality standard. The balancing loop represents the resistance to raising quality standards caused by an increasing quality gap. The lower reinforcing loop represents a second benefit of managing quality standards - the faster completion of the project.

Ford et al. (1993) separated required and voluntary iteration in product development and used a Quality Standard to drive the amount of iteration for quality, while keeping rework not optional. Higher Quality Standards resulted in longer cycle times, depicting a traditional perspective of a quality-for-time tradeoff faced by project managers. Fiddaman, Oliva and Aranda (1993) explored the evolution of the customer requirements using two of Kano's requirement types and a noise factor.
2.5.1.6 The Scope Structure

The final feedback loop is the scope structure. It represents the adjustment of project size (Figure 2-9). Examples of scope reduction adjustments which are not quality adjustments are narrowing the market and therefore product performance requirements and deleting design for manufacturability.

![Diagram of the Scope Structure]

**Figure 2-9: The Scope Structure**

The balancing feedback loop describes the typically-intended impact of scope reduction, to accelerate progress by reducing the amount of work to be done. The reinforcing loop is typically unintended and sometimes goes unrecognized. Less scope may release work that was waiting on the now-deleted work, thereby increasing the work waiting to be done (the Work Queue). Cooper (1980) included adjustments to scope in the shipbuilding model. The dynamics of Fiddaman, Oliva and Aranda's (1993) investigation the growth of project scopes across projects, may influence a single project. Richardson and Pugh (1981) study the impact of when project scope is estimated on cycle time and manpower required.
The six feedback structures that represent existing system dynamics models of product development and the loops which result from their combinations cannot be clearly shown on a single diagram. The complexity of the combination of these loops exceeds the bounded rationality of humans to simulate or predict with any accuracy. This helps explain the difficulty in managing development projects and investigating the impacts of management policies on performance. It also supports the need for computer-based simulation models for investigating coordination.

2.5.2 System Dynamics Models of Product Development Projects

Several researchers have built system dynamics models of product development. Roberts (1974) built a relatively small (30 equations) project model which investigated the management of R&D projects. The primary material which flows through the product development process is generic "job units". Completion of job units is based upon available manpower and productivity. Management decisions are based upon perceived progress, which includes both actual progress and perceptual errors. These decisions include manpower changes, which directly impacts the job unit completion rate.

Cooper (1980) and Reichelt (1990) described the construction and use of large system dynamics models by Pugh-Roberts Associates of large scale shipbuilding operations for claims settlement. Cooper's model included product development in each of the two projects modeled. This model focused upon rework caused by customer changes. Manpower was a primary cost driver and therefore key to the model structure. The model simulates the major phases of shipbuilding operation. Cooper modeled (1980) and subsequently elaborated on (1994, 1993a,b,c) the impacts of rework in projects on cycle time. The model simulates the initial completion of development tasks (basework) and corrective action (rework). A delay in discovering defects slows the completion of unflawed development tasks. The process structure propagates change across project phases with interdependent schedules and disruptions which reduce quality and require rework. Reichelt (1990) describes the dependence of downstream project activities on the
completion of upstream activities in a two-stage process. Cooper and Reichelt's work adds three valuable concepts: 1) the ability of customers to influence cycle time and increase coordination needs in product development processes, 2) the distinction between direct (first-order) and indirect (higher-order) impacts and 3) competition for resources among product development activities.

Richardson and Pugh (1981) developed and explained in detail a model focusing upon the management of single R&D projects. Richardson and Pugh use rework to expand on the Resource Effectiveness portion of the fundamental structure used by Roberts. They distinguish between work performed satisfactorily and tasks requiring rework. Both satisfactory tasks and rework are modeled explicitly. This allows the incorporation of new and potentially important influences on cycle time and coordination such as the error rate in development activities and the rate at which errors are discovered. Richardson and Pugh identify the impractical nature of some cycle time reduction policies such as assuming no rework or assuming a constant project scope. They use their model to illustrate the effects of different assumptions and policies on cycle time.

Abdel-Hamid (1984) modeled software development to better understand project management in light of cost overruns, late deliveries, and user dissatisfaction. The model simulated software production as influenced by human resources management, planning, and controlling. In this model schedule pressure influences resource quantity through the prediction of the work force size necessary to complete the project on schedule. The model's schedule pressure influences resource effectiveness through productivity, the error generation rate, and worker allocation to quality assurance.

Jessen (1990) investigated the behavior and impacts of project manager motivations with a model based upon resources, rework, targets and resource strategy. This model focused on the roles of goal seeking (balancing) feedback loops (pg. 250) in projects. It expands the description of the motivational structures in projects.
Homer et al. (1993) modeled project process structure explicitly by introducing "gate functions" to describe the constraints on work progress imposed by both preceding phases and the work within phases. This model uses graphical table functions to describe these precedence relationships in more detail than possible with the Critical Path or PERT methods. For example two phases can be described as very tightly coupled with the upstream phase limiting the work available throughout the duration of the downstream phase, not just limiting the start or finish of work as in the Critical Path method. Homer et al.'s model uses both available work and resources to constrain progress. The development process structure in the model described in this work has its foundation in the Homer et al. model.

Seville and Kim (1993) built a model based on Kim's earlier model (1988) of product development at a computer hardware company. These models simulated the flow of product development tasks through two stages: product design and process design. Seville and Kim use different levels of coordination to differentiate between "lean production" and "mass production" development organizations (as in Womack et al., 1990). They model coordination between product and process engineers with an exogenous Coordination Fraction decision. Seville and Kim contributed an explicit structure for modeling the coordination effects on resource quantity and effectiveness. They also used a two-stage aging structure and modeled the impacts of factors such as productivity to each stage separately.

Ford et al. (1993) studied the interface between two product development groups within an electronic entertainment equipment manufacturer (Ford and Paynting, 1995). The model focused on the relationships among coordination, schedule, and quality. They explicitly modeled required rework due to errors and optional iteration to meet a quality goal. This allowed them to incorporate the influences of schedule pressure on decisions about iteration for quality. This illustrates modeling cycle time reduction as a tradeoff
between time and project quality. Ford et al. add a distinction between required and voluntary iteration in product development.

2.5.3 Evaluation of System Dynamics Models of Product Development Projects

The existing system dynamics literature has a rich history of modeling development projects. All these models contribute to the description and documentation of the tight linkage between development resources, resource management, and project performance. Many of these structures have been tested and applied adequately to be used as building blocks in the current work. But the current research has several important deficits.

First, the literature rarely addresses the development process directly or how it impacts project behavior. As a specific example of a deficit in the current research some features of static models of development such as the inter-phase precedence relationships used in the Critical Path Method or alternative structures have not been adequately incorporated into dynamic models of product development. Several researchers indirectly describe the tight linkage of development process structure and project performance. But the available research does not explicitly describe the linkages between development processes, policies, and their impacts, i.e. no one has proposed and tested *how* process structure and coordination impact performance. This is partially because the assumptions and specific process structures used to model the development process in the few models which include process structures are not available. The current research addresses these deficits by explicitly modeling development processes and their impacts on project performance.

Second, no current model incorporates all the significant structures developed for other systems and which apply to projects into a single model and tested their combined impacts on project behavior. This research develops such a model.

Third, current system dynamics models of projects model only a few specific project phases. No current model can be easily altered to describe many different types of
projects with different numbers of phases and relationships among phases. This research develops such a model.

2.6 Summary of Literature Evaluation

The product development literature documents the shift from a sequential functional development paradigm to a more concurrent cross-functional team paradigm. It identifies the increased impact of dynamic project features such as increased feedback through many iterations and time delays. It also identifies the cross functional team as a primary tool for improved coordination for improved project performance. However the product development literature does not address in depth the cause-effect relationships within individual projects which link dynamic features to performance. The system dynamics literature investigates in depth development resource structures and their impacts on performance. But little work has been done to explicitly model the development process structures and their impact on project performance. These two deficits are addressed by the current work.
Chapter 3

Model Description

3.1 Introduction

Painting a complete picture of the Product Development Project Model requires descriptions from several perspectives. In this chapter I begin this process by depicting the model from three relatively context-free vantage points. First the model itself is framed by defining its boundaries and level of aggregation. In the second and largest portion of this chapter the model's inner structure is depicted in increasing detail by describing phases, subsystems, and sectors. A table of the foundations of the important model structures completes the model structure section. An initial description and investigation of model behavior is the third vantage point. The sensitivity tests identify the parameters which deserve special attention as the model is applied to specific contexts.

Other perspectives are applied to expand the description and investigation of the model in subsequent chapters. A signal processing model of a portion of the Product Development Project Model is described and used to illustrate an alternative modeling approach in chapter 4. The Python Development Project forms a specific context for the application of the model and its use for policy analysis in chapter 5.
3.2 Model Boundary and Level of Aggregation

The model's scope and focus are reflected in the model boundary. Figure 3-1 delineates the primary features included (endogenous), assumed (exoogenous), and excluded (ignored) from the Product Development Project Model. Among the most important model boundaries are the edges of a single development project. This focuses the research on the inner workings of development projects. While the interaction of projects in a multiple-project development environment can be important (Wheelwright and Clark, 1992; Wheelwright and Sasser, 1991) an improved understanding of the structure and behavior of single development projects is needed as a basis for investigating multiple projects. Such as single project model can be replicated to build a multiple-project model to investigate project interactions.

A second important boundary assumption is a stable development environment, process, and organization throughout the project life. An example of an assumption about a stable development process is the use of exogenous constants to describe the average duration required to complete development activities. These values and function do not change during the simulation. The potential impacts of relaxing the boundary assumptions are discussed in the conclusions.
Figure 3-1: Product Development Project Model Boundary Diagram

Within the model boundary the level of aggregation focuses the research and model purpose. For example the model simulates multiple interdependent development phases within a project. Phases are defined around similar development activities such as product definition, design, testing, and installation. Examples of a single development phase include the preparation of construction drawings in a real estate development project, the writing of software code and the testing of product prototypes.

Another important level of aggregation assumption concerns the fundamental units which flow through projects. I assume that these units are "development tasks". Conceptually a development task is an atomic unit of development work. Examples of development tasks might include the selection of a pump, writing a line of computer code and installing a steel beam. The unit of work used to describe a development task may differ among project phases. For example a product definition phase might use product specifications as the basis for tasks whereas the design phase of the same project might use lines of computer code. Tasks are assumed to be uniform in size and fungible. This assumption
becomes more accurate as task size decreases. Therefore relatively small pieces of development work are selected as tasks. Fungibility is an inherent characteristic of some development tasks (e.g. the delivery and placement of soils for a roadbed). Many other development phases have interdependent but fungible tasks (e.g. software code as in Abdel-Hamid, 1984). The Product Development Project Model provides for the description of task dependencies both within and among phases, as described subsequently in the Development Tasks sector. Tasks are also assumed to be small enough to be flawed or correct but not partially flawed. This assumption also becomes more accurate as task size becomes smaller. These assumptions concerning tasks help identify divisions among development phases and development tasks.

I have disaggregated development within each phase into four activities: basework, quality assurance, rework and coordination. Basework is the completion of a development task the first time. Subsequent completions which are required to correct flaws or iterate for quality are referred to as Rework. Rework includes all forms of iteration regardless of cause. The search for flaws is Quality Assurance (QA) or Inspection. Flaws include errors which must be corrected for product functionality and optional improvements for quality. Coordination is the integration of the product development project among phases due to releasing and inheriting flawed tasks from other phases.

Resources for each phase have been aggregated into a single labor type. This reflects an assumption that other development resources such as testing machines and administrative support are used in proportion to development labor. The primary model assumptions concerning the level of aggregation are listed below.

• Development projects consist of a network of dependent phases under a project management structure.

• The progress of a development phase can be reflected in the flow of development tasks through and among development phases.

• Development tasks are small, uniform in size and either flawed or correct but not partially flawed.
• Development occurs through four activities: basework, quality assurance, rework, and coordination.

• The repair of flawed tasks for basic product functionality and optional iteration for quality have similar characteristics and can be modeled together as Rework.

• Different resource types can be aggregated into a single labor group for each development phase.

The Product development Project Model’s boundary and level of aggregation focus the research. The internal structure of the model which simulates a development project is described in the next section.

3.3 Model Structure

3.3.1 Introduction

Operationally the model is a set of nonlinear ordinary differential equations. Appendix 3.1: Model Equations provides a complete listing of those equations. The equations are arrayed to allow the simulation of a flexible number of development phases and include many equations to manage the modeling of multiple phases. This results in a high number of model parameters and relationships in a network which is too complex to illustrate with multiple phase diagrams. However diagrams of a single phase will be used for model description purposes with explanations of deviations for multiple phases. Definitions of the model parameters used in the model equations are given in Appendix 3.2: Model Parameters.

The model consists of a set of interrelated development phases and a set of project management features. Each phase is customized to reflect a specific stage of product development. A phase dependency network describes the forward flow of work through the project. Figure 3-2 shows a simple phase dependency network for a real estate development project.
Figure 3-2: A Phase Dependency Network for a Development Project

Project phases are linked in several ways:

- Work flows in upstream phases constrain progress in dependent downstream phases. The locations of these constraints are shown by the arrows in the project's phase dependency network (Figure 3-1).
- Errors inherited by downstream phases from upstream phases corrupt downstream work.
- Inherited errors that are discovered in downstream phases are returned for correction to the phase where the error was generated.
- Coordination with other phases is required by discovering inherited errors or having errors generated within a phase discovered by downstream phases.
- Completion and expected completion dates of phases influence the project deadline. The project deadline in turn influences phase deadlines.
- Poor schedule, quality, and cost performance in any phase increases the impacts of non-conformance to the project targets. Those project level impacts influence individual phase targets.

The basis for the model structure is described below and summarized in Table 3-2, located after the model subsystem descriptions.

3.3.2 Model Subsystems

The model is relatively large with approximately twenty five stocks for each development phase and five project management stocks. For descriptive purposes the model has been disaggregated into five subsystems (Figure 3-3): process structure, scope, resources, targets, and performance. Subsystems have been further disaggregated into sectors.
Subsystems and sectors are tightly linked through shared parameters. For clarity these parameters are shown in each sector diagram where the parameter is used.

**Figure 3-3: Model Subsystems**

### 3.3.3 The Process Structure and Scope Subsystems

#### 3.3.3.1 Introduction

Impacts of the development process and the amount of project work on performance are modeled with the Process Structure and Scope subsystems. The Development Tasks subsystem describes the nature of the development process, while the Scope subsystem simulates the original project scope and increases due to rework. These subsystems include the Development Tasks, Internal Errors, Upstream Errors and Downstream Errors sectors.

#### 3.3.3.2 The Development Task Sector

The process structure portion of the model is one of the most important contributions of this research to model structure. The Development Task sector is the core of how the
Product Development Project Model describes development processes. A diagram of the complete Development Tasks Sector is shown in Figure 3-15. One of the most important interactions of development process and resources occurs at the four development activities in the Development Tasks sector. Each activity proceeds at the minimum pace allowed by its process and resources, as modeled with the following equations:

\[
\text{QA\_inspection\_rate(Phase)} = \text{MIN(QA\_Process\_Limit(Phase),QA\_Labor\_Limit(Phase))}
\]

\[
\text{Rework(Phase)} = \text{MIN(RW\_Process\_Limit(Phase),RW\_Labor\_Limit(Phase))}
\]

\[
\text{Basework(Phase)} = \text{MIN(BW\_Process\_Limit(Phase),BW\_Labor\_Limit(Phase))}
\]

\[
\text{Coordination(Phase)} = \text{MIN(Coord\_Process\_Limit(Phase),Coord\_Labor\_Limit(Phase))}
\]

Figure 3-4 illustrates the two feedback loops which are the basis for the process structure. The balancing loop depicts the reduction in the number of tasks available for Basework as work is completed. In this loop the Basework rate is based on the Tasks Available for Basework and the Minimum Basework Duration. An increase in the Basework rate increases the number of Tasks Completed, which decreases the number of Tasks Available for Basework, which reduces the Basework rate. This loop introduces the first of two types of parameters used to describe the development process in a phase, the Minimum Activity Duration. The Minimum Activity Duration is the average time required to complete a task if all required information, materials and resources are available and no flaws are generated. It describes the purest time constraint which the process imposes on progress by answering the question "How fast on average can a task be completed if everything needed is available?" All four development activities (basework, quality assurance, rework and coordination) apply this concept. This allows more detailed and accurate descriptions of a development process than by modeling a single development activity. In Figure 3-4 the Minimum Activity Duration is applied to basework. Feedback loops similar to the balancing loop shown in Figure 3-4 are used to describe the role of the Minimum Quality Assurance Duration in discovering flawed tasks, the Minimum Rework Duration in correcting flawed tasks, and the Minimum Coordination Duration in integrating development project phases.
The reinforcing loop shown in Figure 3-4 models the increase in the number of tasks which are available to the basework activity due to the completion of work. In this loop an increased basework rate raises the number of Tasks Completed, which raises the total number of tasks which can be completed. The total number of tasks which can be completed includes both tasks which have been completed and tasks which are available and waiting to be completed. This quantity of tasks is also dependent on the nature of the development process as described by the process's Internal Precedence Relationship. Increasing the number of Tasks Completed & Waiting to be Completed raises the Tasks Available for Basework and thereby further raises the Basework rate. The reinforcing loop introduces the second type of descriptor of specific development processes, the Internal Precedence Relationship. Internal Precedence Relationships describe the availability of work based solely on the amount of work which has been completed.
Erecting structural steel for a ten story building one story at a time from the ground up provides an example of an Internal Precedence Relationship. Initially 0% is Completed and only the first floor (10%) is Completed or Available to be Completed. When the first floor of steel is erected 10% is Completed and the second floor (another 10%) becomes available, making 20% Completed or Waiting to be Completed. This linear progression continues until the completion of the ninth floor (90% Completed) releases the final floor for completion (100% Completed or Waiting to be Completed). A graphic function which describes this Internal Precedence Relationship is shown in Figure 3-5. The Internal Precedence Relationship describes the available-work constraint which a development process imposes on itself by answering the question "How much work can be completed based upon how much work has been completed?"

![Figure 3-5: A Linear Internal Precedence Relationship](image)

An advantage of the system dynamics methodology and the model structure used here is the ability to describe nonlinear Internal Precedence Relationships (Graham 1980). This allows varying degrees of concurrent development within a single phase can be described with Internal Precedence Relationships by altering the shape of the curve in the graphical function. The Internal Precedence Relationship example shown in Figure 3-6 is based upon the design of code for the development of a computer chip. Documentation for the shape of the curve is provided in a later chapter. Initially a few very important blocks of
code must be designed. This is the reason for the flat portion of the curve on the left side of the function. Their completion makes the design of many more blocks possible. This is the reasoning behind increasing rate of availability in the left portion of the curve. The increase in available work slows as the design nears completion and the blocks of code must be integrated. This produces the flat "tail" of the curve.

![Figure 3-6: A Nonlinear Internal Precedence Relationship](image)

A development process's Internal Precedence Relationship describes the available work constraints which the tasks aggregated into a single phase impose on each other. It estimates the impacts of the dependency network which exists among the phase's tasks. For example if the erection of structural steel for a high-rise building was the phase the Internal Precedence Relationship would reflect that columns must be installed before the beams which they support. These constraints can act as a bottleneck in the availability of work. Most previously published system dynamics models of projects have assumed that all uncompleted tasks are available for completion (e.g. Abdel-Hamid, 1984; Richardson and Pugh, 1981; Roberts, 1974). This assumption implies that all tasks are independent and could be performed in parallel and that the nature of the development process imposes no constraints on the number of tasks available for completion. However product development research (e.g. Rosenthal, 1992; Clark and Fujimoto, 1991) and the steel
erection example above show that processes can and frequently do internally constrain the availability of work.

The stock and flow structure of the development process within a single phase follows from the feedback structure shown in Figure 3-4. Development tasks flow into and through three stocks: the Completed not Checked, Known Rework, and Checked & Released stocks (Figure 3-7).

![Diagram](image)

**Figure 3-7: Process Structure Stocks and Flows (Single Phase)**

Tasks are completed for the first time through the performance of basework. They accumulate in the Completed not Checked stock. If no tasks are flawed or those flaws are not found tasks leave the Completed not Checked stock and pass through the Release Tasks flow into the Checked & Released stock. This represents delivering tasks to the managers of downstream phases or to customers. Flawed tasks are modeled in the Errors sectors. Tasks which are found to be flawed flow to the Known Rework stock. The Rework flow returns corrected tasks to the Completed, not Checked stock for inspection. The following stock equations describe these accumulations explicitly.

\[
\text{Tasks}_{\text{Completed}}(\text{Phase}) = \text{Tasks}_{\text{Completed}}(\text{Phase}) + dt*(\text{Basework}(\text{Phase}) + \text{Rework}(\text{Phase}) - \text{Release}_\text{Tasks}(\text{Phase}) - \text{RW}_{\text{due to InPhase QA}}(\text{Phase}))
\]

\[
\text{Known}_{\text{Rework}}(\text{Phase}) = \text{Known}_{\text{Rework}}(\text{Phase}) + dt*(\text{RW}_{\text{due to InPhase QA}}(\text{Phase}) - \text{Rework}(\text{Phase}))
\]
Tasks\_Released(Phase) = Tasks\_Released(Phase) + dt*\(\text{Release\_Tasks(Phase)}\)

Figure 3-8 shows the process structure for a single phase with auxiliary parameters.

![Process Structure Diagram]

**Figure 3-8: Process Structure (Single Phase)**

The form taken by the limits on progress imposed by the development process structure for each of the four development activities are similar. A phase's demand for each activity (Basework, Quality Assurance, Rework and Coordination) is the number of tasks available for the activity. The general process limit equation is:

\[
\text{Development Activity Process Limit} = \frac{\text{Tasks Available for the Activity}}{\text{Minimum Activity Duration}}
\]

Minimum Activity Durations describe the relative difficulty of the four development activities. The structures which describe the development activities will be described in increasing complexity from Rework to Coordination to Quality Assurance to Basework.
The Rework flow structure is the simplest. The value of the Tasks Available for the Activity parameter in the equation above for the Rework flow is the number of tasks in the Known Rework stock. The formulation assumes that all tasks in the rework stock are independent. The Rework process limit equation is:

\[
\text{Rework Process Limit} = \frac{\text{Known Rework}}{\text{Minimum Rework Duration}}
\]

Demand for the Coordination activity is the Coordination Backlog. Since coordination is the interaction of developers across phases it is required only for multiple phases. This stock is the accumulation of the sum of the tasks which have been corrupted due to inheriting flawed tasks from upstream and the flawed tasks which have been released and returned by downstream development phases less the tasks which have been coordinated. Figure 3-15 shows this structure. The Coordination process limit equation is:

\[
\text{Coordination Process Limit} = \frac{\text{Coordination Backlog}}{\text{Minimum Coordination Duration}}
\]

The value of the Tasks Available for the Activity parameter for the Quality Assurance rate is the number of tasks which have been completed but not yet checked. This is the Completed not Checked stock. Therefore the Quality Assurance process limit equation is:

\[
\text{Quality Assurance Process Limit} = \frac{\text{Completed not Checked}}{\text{Minimum QA Duration}}
\]

Quality Assurance is the basis for two flows. The first flow is the Find Flawed Tasks flow which depends on the Quality Assurance rate and the effectiveness of those efforts at finding flawed tasks. Quality assurance effectiveness is measured with the probability of finding a flawed task. This is the product of the probability of finding a flaw if it exists and the probability of a task being flawed. Therefore the Find Flawed Tasks equation is:

\[
\text{Find Flawed Tasks} = \frac{\text{Completed not Checked}}{\text{Minimum QA Duration}} \ast p(\text{Flaw found if exists}) \ast p(\text{Task is Flawed})
\]
The probability of a flawed task being found if it exists is based upon the adequacy of the quality assurance effort, as measured by the ratio of the quality assurance labor applied to the quality assurance labor required. This is described in the resources sector of the model. The probability that a task is flawed is the ratio of the number of unchecked flawed tasks (in the Internal Errors sector) to the total number of unchecked tasks.

The second flow driven by Quality Assurance is the Release Tasks flow. All tasks which are checked leave the Completed not Checked stock. Those that are found to be flawed are sent to the Known Rework stock as described above. Tasks which are found to be unflawed (those without flaws and those with flaws that were missed) are released. Therefore the equation for the process limit on the Release Tasks flow is the total number of tasks checked less those found to have flaws:

\[
\text{Release Tasks} = \text{Quality Assurance rate - Find Flawed Tasks}
\]

The structure of the Basework flow is the most complex of the four development activities. The demand for Basework is the total number of tasks which \textit{can} be completed less the tasks which \textit{have already} been completed at least once. Therefore the Basework process limit equation is:

\[
\text{Basework Process Limit} = \\
\frac{(\text{Total Tasks Available - Completed not Checked - Checked \& Released - Known Rework})}{\text{Minimum Basework Duration}}
\]

The Total Tasks Available is the number of tasks which could be completed based upon the tasks which have been completed and released. This constraint is described with the phase's Internal Precedence Relationship and answers the question "What percent of the tasks are available for initial completion based upon the percent which have been completed and released?" Known Rework is not included in the Total Tasks Available because flawed tasks do not make additional work available. The equation for the Total Tasks Available is:
Total Tasks Available = Number of Tasks in Project Scope * 
Min(Internal Precedence Relationship, External Precedence 
Relationship)

The Internal and External Precedence Relationships are important characterizations of the nature of specific development process. They can be nonlinear in nature and is therefore described with graphic functions.

The Development Task sector includes two additional flows and one additional stock for modeling inter-phase linkages (Figure 3-15). The first of these flows to be described models the corruption of completed work in a focal phase due to inheriting flawed tasks from upstream.

\[
RW\_due\_to\_Corrupted\_tasks(Phase)=\text{(Net\_Corrupted\_and\_Found\_Tasks(Phase)}-\text{(QA\_inspection\_rate(Phase)}*\text{(Net\_Corrupted\_and\_Found\_Tasks(Phase)/} \text{(QA\_inspection\_rate(Phase)+1e-9)}*\text{prob\_Task\_Flawed\_and\_Found(Phase)})))
\]

Rework due to Corrupted Tasks moves tasks which are completed but not checked into the Known Rework stock based on the fraction of tasks found to be corrupted by inherited errors (described later). Relative sizes of the phases are used to scale upstream errors found into tasks corrupted in the focal phase.

The second flow required for multiple phases is Rework due to Errors Discovered by Downstream Phases.

\[
RW\_due\_to\_Dwnstrm\_QA(Phase)=\text{Total\_Err\_disc\_by\_Dn(Phase)}+\text{Total\_Corrupted\_by\_Upstream\_Retraction(Phase)}
\]

Errors which are discovered downstream are aggregated with released work corrupted by upstream errors (described later), removed from the Tasks Released stock and added to the Known Rework stock. The revised Development Task sector stock equations which include these inter-phase flows are:
Tasks_Completed(Phase) = Tasks_Completed(Phase) + dt*(Basework(Phase) + Rework(Phase) - Release_Tasks(Phase) - RW_due_to_InPhase_QA(Phase) - RW_due_to_Corrupted_tasks(Phase))

Known_Rework(Phase) = Known_Rework(Phase) + dt*(RW_due_to_InPhase_QA(Phase) + RW_due_to_Corrupted_tasks(Phase) + RW_due_to__Dwnstrm_QA(Phase) - Rework(Phase))

Tasks_Released(Phase) = Tasks_Released(Phase) + dt*(Release_Tasks(Phase) - RW_due_to__Dwnstrm_QA(Phase))

The two inter-phase error flows control the model's fourth development activity, coordination, and determine the effort required to address inherited errors and released discovered errors. Each of the two flows described above model development activities which require interaction between development phases. They generate a need for coordination.

Current_Coord_added(Phase) = RW_due_to__Dwnstrm_QA(Phase) + RW_due_to_Corrupted_tasks(Phase)

The accumulation of these flows represents a backlog of coordination work needing to be completed. The performance of the coordination activity reduces the coordination backlog. The following equation describes the accumulation of coordination work.

Coord.Backlog(Phase) = Coord.Backlog(Phase) + dt* (Current_Coord_added(Phase) - Coordination(Phase)) .

The Development Tasks sector also uses a Fraction Available due to External Gates parameter to model available-work constraints between phases (inter-phase constraints). Tasks Available for Basework is based on the minimum of the internal and external gates. The External Precedence Relationships describe the available-work constraints between development phases in a manner analogous to the internal available-work constraint described by the Internal Precedence Relationships. An External Precedence Relationship describes the availability of work in a downstream phase based on the amount of work which has been released by an upstream phase. The input (abscissa) of an External
Precedence Relationship is the Percent of Upstream Tasks Released. The output is the Percent of Downstream Tasks Available for Basework.

Like a development phase's Internal Precedence Relationship, an External Precedence Relationship between two development phases can act as a bottleneck in the availability of work. Most previously published system dynamics models of projects have assumed that all uncompleted tasks are available for completion (e.g. Abdel-Hamid and Madnick, 1991; Richardson and Pugh, 1981; Roberts, 1974). This assumption implies that the nature of the development process imposes no constraints on the number of tasks available for completion. However the success of the Critical Path and PERT methods in statically modeling inter-phase dependencies in development projects and product development research (e.g. Rosenthal, 1992; Clark and Fujimoto, 1991; Eppinger et al., 1990) show that relationships among development phases can and often do constrain the availability of work.

The purpose of External Precedence Relationships is the same as the precedence relationships used in the Critical Path and PERT methods: to describe the dependencies of development phases on each other for the initial completion of work. However there are several important differences between External Precedence Relationships and precedences used in the Critical Path and PERT methods.

- External Precedence Relationships describe the dependency between two phases along the entire duration of the phases instead of only at the start and finish of the phases as in the Critical Path and PERT methods.
- External Precedence Relationships can be nonlinear.
- External Precedence Relationships describe a dynamic relationship between development phases by allowing the output (Percent Tasks Available for Basework) to fluctuate over the life of the project depending on the current conditions of the project, as described by the External Precedence Relationship's input (Percent Upstream Tasks Released).

External Precedence Relationships can be used to describe rich inter-phase relationships which cannot be described with Critical Path and PERT precedences. For example a downstream phase which is constrained by the release of upstream tasks throughout its
duration (not only at the beginning or end of the phase) in a linear relationship can be described with a "lockstep" External Precedence Relationship, as shown in Figure 3-13.

![Figure 3-9: A "Lockstep" Constraint described with An External Precedence Relationship](image)

The inter-phase relationship in Figure 3-9 is linear. One of the advantages of the model structure used here is the ability to describe nonlinear inter-phase constraints. Varying levels of concurrence in the development process can be described with External Precedence Relationships by altering the shape of the curve in the graphical function. Infinite order delays between phases can also be described by shifting the point along the abscissa at which the first downstream tasks become available. As an example the External Precedence Relationship shown in Figure 3-10 describes an inter-phase relationship in which the downstream phase must wait until the upstream phase has released 20% of its tasks and then can perform Basework relatively concurrently until near the completion of the downstream phase.
Figure 3-10: A Delayed Concurrent Constraint described with An External Precedence Relationship

External Precedence Relationships are used in the Product Development Project Model to describe some of the complexity of the inner structure of a product development project and its impacts on project performance.

External Precedence Relationships reflect the concurrence of phases which are dependent.

Concurrence(up,down)=FIFZE(1.00,TABHL(T6(*,up,down),
Fraction_Released(up),0,1,0.10),Dependency(up,down))

Fraction_Avail_due_to__Ext_gates(Phase)=MIN(FIFZE(1.00,Concurrence(1,Phase),Dependency(1,Phase)),FIFZE(1.00,Concurrence(2,Phase),Dependency(2,Phase)),FIFZE(1.00,Concurrence(3,Phase),Dependency(3,Phase)),FIFZE(1.00,Concurrence(4,Phase),Dependency(4,Phase)),FIFZE(1.00,Concurrence(5,Phase),Dependency(5,Phase)))

Fraction_Released(Phase)=Tasks_Released(Phase)/Task_List(Phase)

The Concurrence variable reflects the External Precedence Relationship as discussed above, using the Fraction Released by the upstream phase to constrain the work available in the downstream phase. The Fraction Available due to External Gates variable includes only the dependent upstream phases in determining the available work with a set of
switches linked to the project network. The switches and network are linked with the Dependency variable, which is 1 if the phases are dependent and 0 if they are not.

The complete Development Tasks sector is shown in Figure 3-11.

![Diagram of Development Tasks Sector]

Figure 3-11: The Development Tasks Sector

3.3.3.3 The Errors Sectors

Three sectors model errors: the internal, inherited, and released errors sectors. Errors generated, discovered, and corrected within a single phase are modeled by the Internal Errors sector. A co-flow structure (Homer, 1983 Appendix Q) is used in which the stocks
and flows are directly related to the stocks and flows in the Development Tasks sector. A comparison of Figure 3-11 and Figure 3-12 below shows their similarity. Flawed tasks are discovered through the Quality Assurance activity. As described previously tasks found to be flawed move through the Find Flawed Tasks flow from the Completed not Checked stock to a stock of Known Rework. These tasks are corrected through the Rework Tasks activity and returned to the Completed not Checked stock. The Internal Errors sector models the generation of flaws, which can be generated during both Basework and Rework. This means that a task being reworked to correct an existing defect may become flawed during the rework process. Because quality assurance efforts are not perfect some flaws are missed (i.e. Type 2 errors are allowed). Therefore some flawed tasks are mistakenly considered to be unflawed and are released with the unflawed tasks. These errors are inherited by the phase's dependent downstream phases. The model assumes that unflawed tasks mistakenly considered flawed are incorporated into the values of the Minimum Activity Durations.

Figure 3-12: The Internal Errors Sector
The key equations used to model the internal errors sector are described below. Several of the tables used in the Product Development Project Model to describe nonlinear relationships between parameters have an "S" shape. These curves have upper and lower limits to output values, small unit changes in output near those limits and larger unit changes in output near the center of the input range. The reasoning behind this curve follows. One limit is at or near normal operating condition (e.g. labor provided approximates labor needed). Changes in input values near this limit generate no or small changes because developers do not perceive enough digression from normal conditions to trigger a significant response. Developer responses and output values change more as input conditions move further from normal and developers notice and react to the variance. Output unit changes decrease again as input values approach the other limit based on the assumption of smooth continuous developer response to limits.

\[ \text{Generate Errors(Phase)} = (\text{Basework(Phase)} + \text{Rework(Phase)}) \times \text{prob.Task_flawed(Phase)} \]

\[ \text{prcbTask_flawed(Phase)} = 1 - ((1 - \text{Basic_prob_flawed_Task(Phase)}) \times (1 - \text{prob_of_err_gen_by_QofP(Phase)})) \]

Errors are generated during the basework and rework development activities. The probability of error generation is based on two factors which combine to cause errors. The inherent complexity of the task is reflected in the basic probability that a task is flawed. The impacts of the development work are reflected in the probability of an error being generated by the quality of practice. Each of these probabilities are used to find the probability of no error being generated by the task complexity or quality of practice. These "clean" probabilities are combined to find the probability of a task being flawed by neither of these factors. The resulting probability of no error is used to find the probability of an error by subtracting from 1.

\[ \text{prob_of_err_gen_by_QofP(Phase)} = \text{TABHL(T3,Quality_of_Practice(Phase)/Ref_Qual_of_Practice(Phase),0.1,0.10)} \]

\[ T \times T3 = 0.55/0.45/0.36/0.28/0.21/0.15/0.10/0.06/0.03/0.01/0.00 \]

The quality of practice influences the probability of error generation through a reverse "S" shaped curve which does not increase errors if the quality of practice is above a
reference level. Excess quality of practice is assumed to not hurt a project. The curve rises to a maximum of 55% when the quality of practice is zero. This assumes that there is a limit to the harmful impacts of poor quality of practice on the generation of errors. This is based on the assumption that there is some minimum underlying ability in the developers to perform development tasks which cannot be totally eroded by the conditions of the project. This is a reasonable assumption when developers are professionally trained and the process interacts using Mintzberg's (1979) standardization of skills.

Several of the stocks and flows in the internal errors sector are directly analogous to stocks and flows in the development tasks sector. The error parameters differ from the task parameters due to the densities of flaws. The following equations are used to model those densities.

\[
\text{Compl\_Task\_error\_density(Phase)} = \frac{\text{Our\_Undiscd\_Errors(Phase)}}{\text{Tasks\_Completed(Phase)}}
\]

\[
\text{Our\_Discd\_Error\_density(Phase)} = \frac{\text{Our\_Discd\_Errors(Phase)}}{\text{Known\_Rework(Phase)} + 1e^{-9}}
\]

\[
\text{Rel\_Task\_error\_density(Phase)} = \frac{\text{Our\_Errors\_Released(Phase)}}{\text{Tasks\_Released(Phase)} + 1e^{-9}}
\]

\[
\text{Clean\_Task\_error\_density(Phase)} = (\text{Compl\_Task\_error\_density(Phase)} \times (1 - \text{Prob\_finding\_our\_error\_if\_exists(Phase)})) / ((1 - \text{Compl\_Task\_error\_density(Phase)}) + (\text{Compl\_Task\_error\_density(Phase)} \times (1 - \text{Prob\_finding\_our\_error\_if\_exists(Phase)})) + 1e^{-9})
\]

This last density is the number of flawless tasks divided by the sum of the number flawed and flawed but missed tasks. The numerator is the product of the probability of a task being flawless and the probability of finding a flaw if it exists. The denominator is the numerator plus the compliment of the probability of a task being flawed.

The error coflow flow equations are formed by combining the densities and the task flows. The error coflow stock equations are the accumulations of the flows.

\[
\text{Our\_Undiscd\_Errors(Phase)} = \text{Our\_Undiscd\_Errors(Phase)} + dt \times (\text{Generate\_Errors(Phase)} - \text{Release\_Errors(Phase)} - \text{Disc\_Our\_Errors(Phase)} - \text{Errors\_lost\_in\_Corrupted\_Tasks(Phase)})
\]

\[
\text{Disc\_Our\_Errors(Phase)} = \text{RW\_due\_to\_InPhase\_QA(Phase)}
\]
Errors_lost_in_Corrupted_Tasks(Phase)=Compl_Task_error_density(Phase) * 
RW_due_to_Corrupted_tasks(Phase)

Release_Errors(Phase)=(Release_Tasks(Phase)*Clean_Task_error_density(Phase))
Our_Errors_Released(Phase)=Our_Errors_Released(Phase)+dt * 
(Release_Errors(Phase)-Receive_Our_Errors_fr_Dn(Phase))

Receive_Our_Errors_fr_Dn(Phase)=Total_Err_disc_by_Dn(Phase)+ 
Total_Corrupted_by_Upstream_Retraction(Phase)

Our_Discd_Errors(Phase)=Our_Discd_Errors(Phase)+dt*(Receive_Our_Errors_fr_Dn(Phase)+ 
Disc_Our_Errors(Phase)-Correct_Our_Errors(Phase))

Correct_Our_Errors(Phase)=Rework(Phase)*Our_Discd_Error_density(Phase)

Total_Err_disc_by_Dn(Phase)=Up_Flawed_Tasks_found(Phase,1)+ 
Up_Flawed_Tasks_found(Phase,2)+Up_Flawed_Tasks_found(Phase,3)+ 
Up_Flawed_Tasks_found(Phase,4)+Up_Flawed_Tasks_found(Phase,5)

The total number of errors returned to a phase is the sum of the errors released by that 
phase and discovered by all the downstream phases.

Prob_finding_our_error_if_exists(Phase)=TABHL(T2,QA_Status(Phase),0.20,2.0)
T T2=0.00/0.335/0.535/0.685/0.81/0.88/0.925/0.96/0.985/1.00/1.00

The probability of finding an existing error is based on the adequacy of the quality of 
practice. No errors can be found if the quality of practice is zero. This assumes that the 
project conditions can degrade the quality of the work done by the developers to such a 
degree that they miss all the errors in the work they inspect. This is a reasonable lower 
bound. The probability of finding errors based on the adequacy of the actual quality of 
practice increases as the actual quality of practice rises above a reference value. An upper 
bound of finding all errors (assuming other factors do not prevent discovery) is 
approached as the quality of practice reaches 18 times the reference value.

prob_Task_flawed_and_Found(Phase)=Prob_finding_our_error_if_exists(Phase) * 
Compl_Task_error_density(Phase)

The probability that a task is both flawed and discovered to be flawed is the product of 
the probabilities that it is flawed and the probabilities that the flaw is found.
Errors received by a phase from an upstream phase are modeled with the Upstream Errors sector. Figure 3-13 shows an example of this structure for a focal phase with two upstream phases. The Product Development Project Model can model the inheritance of errors from a flexible number of phases. These errors corrupt tasks done in the focal phase. Each phase discovers a portion of its tasks which have been corrupted by the errors it inherits based on the quantity and effectiveness of its quality assurance efforts. Multiple corruptions of the same task are eliminated and the net corrupted tasks are used in the Development Tasks and Internal Errors sectors.

Figure 3-13: The Upstream Errors Sector

The key equations used to model the upstream errors sector are described below.
\[
\text{prob\_Task\_Corrupted\_and\_found}(up,\text{Phase}) = \text{prob\_find\_Up\_error\_if\_exists}(\text{Phase}) \times \\
\text{Rel\_Task\_error\_density}(up) \times \text{Dependency}(up,\text{Phase})
\]

The probability that an inherited task is both flawed and discovered to be flawed is the product of the probabilities that it is flawed and the probabilities that the flaw is found.

\[
\text{Tasks\_Corrupted\_and\_Found}(up,\text{Phase}) = \text{QA\_inspection\_rate}(\text{Phase}) \times \\
\text{prob\_Task\_Corrupted\_and\_found}(up,\text{Phase})
\]

The number of tasks found to be corrupted is the probability that finding corrupted tasks times the rate at which tasks are being inspected in the downstream phase.

\[
\text{Corrupted\_task\_discoveries}(\text{Phase}) = \text{Tasks\_Corrupted\_and\_Found}(1,\text{Phase}) + \\
\text{Tasks\_Corrupted\_and\_Found}(2,\text{Phase}) + \text{Tasks\_Corrupted\_and\_Found}(3,\text{Phase}) + \\
\text{Tasks\_Corrupted\_and\_Found}(4,\text{Phase}) + \text{Tasks\_Corrupted\_and\_Found}(5,\text{Phase})
\]

The total number of tasks discovered to be corrupted is the sum of the corrupted tasks found from all upstream dependent phases. Phases not linked to the focal phase have been assigned zero tasks corrupted. This value can include multiple discoveries of the same flawed task.

\[
\text{percent\_Multiple\_Corruption\_discoveries}(\text{Phase}) = \\
(\text{prob\_Task\_Corrupted\_and\_found}(1,\text{Phase}) \times \text{prob\_Task\_Corrupted\_and\_found}(2,\text{Phase})) + \\
(\text{prob\_Task\_Corrupted\_and\_found}(1,\text{Phase}) \times \text{prob\_Task\_Corrupted\_and\_found}(3,\text{Phase})) + \\
(\text{prob\_Task\_Corrupted\_and\_found}(1,\text{Phase}) \times \text{prob\_Task\_Corrupted\_and\_found}(4,\text{Phase})) + \\
(\text{prob\_Task\_Corrupted\_and\_found}(1,\text{Phase}) \times \text{prob\_Task\_Corrupted\_and\_found}(5,\text{Phase})) + \\
(\text{prob\_Task\_Corrupted\_and\_found}(2,\text{Phase}) \times \text{prob\_Task\_Corrupted\_and\_found}(3,\text{Phase})) + \\
(\text{prob\_Task\_Corrupted\_and\_found}(2,\text{Phase}) \times \text{prob\_Task\_Corrupted\_and\_found}(4,\text{Phase})) + \\
(\text{prob\_Task\_Corrupted\_and\_found}(2,\text{Phase}) \times \text{prob\_Task\_Corrupted\_and\_found}(5,\text{Phase})) + \\
(\text{prob\_Task\_Corrupted\_and\_found}(3,\text{Phase}) \times \text{prob\_Task\_Corrupted\_and\_found}(4,\text{Phase})) + \\
(\text{prob\_Task\_Corrupted\_and\_found}(3,\text{Phase}) \times \text{prob\_Task\_Corrupted\_and\_found}(5,\text{Phase})) + \\
(\text{prob\_Task\_Corrupted\_and\_found}(4,\text{Phase}) \times \text{prob\_Task\_Corrupted\_and\_found}(5,\text{Phase}))
\]

The total probability of a flaw discovery being repeated is the sum of the probabilities of a multiple discovery by each of the possible phase interactions. The probability of a repeated discovery is the product of each of the individual discovery probabilities.

\[
\text{Multiple\_Corruption\_discoveries}(\text{Phase}) = \text{QA\_inspection\_rate}(\text{Phase}) \times \\
\text{percent\_Multiple\_Corruption\_discoveries}(\text{Phase})
\]

Multiple discoveries occur at the rate of inspection times the percent of tasks found more than once to be flawed.
Net_Corrupted_and_Found_Tasks(Phase)=Corrupted_task_discoveries(Phase)-
Multiple_Corruption_discoveries(Phase)

The net number of tasks found to be corrupted is the actual number of corrupted and
flawed tasks corrected for multiple error discoveries. This is the total number of
discoveries less the number of multiple discoveries.

Up_Flawed_Tasks_found(up,Phase)=Tasks_Corrupted_and_Found(up,Phase)*
TaskListScale(up,Phase)

Total_Err_disc_by_Dn(Phase)=Up_Flawed_Tasks_found(Phase,1)+
Up_Flawed_Tasks_found(Phase,2)+Up_Flawed_Tasks_found(Phase,3)+
Up_Flawed_Tasks_found(Phase,4)+Up_Flawed_Tasks_found(Phase,5)

The flawed tasks returned to the upstream phase are those discovered by the downstream
phases increased or decreased by the relative sizes of the upstream phase and each of the
dependent downstream phases. The total returned flawed tasks is the sum of those
return from all downstream phases.

prob_find_Up_error_if_exists(Phase)=MIN(1,Coord_effect_on_Find_Up_Errors(Phase)*
QoFP_Error_Disc_effect(Phase))

Coord_effect_on_Find_Up_Errors(Phase)=TABHL(T1,Coord_Status(Phase),0,1,0.10)
T T1=0.00/0.015/0.045/0.115/0.265/0.54/0.715/0.84/0.92/0.975/1.00

QoFP_Error_Disc_effect(Phase)=TABHL(T4,Quality_of_Practice(Phase)/
Ref_Qual_of_Practice(Phase),0,1,0.10)
T T4=0.20/0.20/0.22/0.25/0.3/.4/.65/.7/.85/.95/1.0

The probability of finding an upstream error if it exists is impacted by the adequacy of
coordination and the level of the quality of practice. Both relationships have an "S" shape.
The lower limit of the coordination relationship is zero, indicating that if there is no
coordination there is no chance of finding upstream errors. The quality of practice
relationship has a lower limit of 0.20, indicating that some errors could be found if two
teams were coordinating but one had a low quality of practice. Both upper limits are 1.0
since finding more than 100% of the existing errors is unreasonable.

Two types of project errors cause tasks to flow from the Tasks Released stock to the
Known Rework stock. The first is errors which are released to downstream phases,
discovered there, and returned for correction. The second type of project error is upstream
errors which have been discovered and require the rework of completed and released tasks in downstream dependent phases. This represents the notification of other phases of an error by developers. This second type of errors reflects the following conditions: the middle phase in a sequential set of three phases completes its tasks before the final phase discovers errors created and released by the first phase and not discovered by the middle phase. The final phase returns those errors to the first phase for correction, generating rework for the first and final phases. Since the middle phase is completed these newly discovered errors will not be reflected in the corruption of tasks in the middle phase. In real projects these errors by the first phase will require the rework of released tasks in the middle phase as well. This relationship models the notification of downstream phases when an upstream phase is notified that it has made an error. The model uses the following equation to model this flow.

\[ RW_{due\ to\ Downstrm\ QA(Phase)} = \text{Total\ Err\ disc\ by\ Dn(Phase)} + \text{Total\ Corrupted\ by\ Upstream\ Retraction(Phase)} \]

The flow is the sum of the flaws due to released errors discovered downstream and the flaws caused by the retraction of finished work upstream due to error discoveries. The two contributors to the sum are the totals of the errors for each of the upstream or downstream phases linked to the focal phase.

### 3.3.4 The Resources Subsystem

The structural components of the Resources subsystem are based on previously constructed and tested system dynamics models which are documented in the literature. Table 3-1 at the end of the model structure description lists the primary model references for each model sector. The Resources subsystem consists of the Gross Labor, Labor Allocation, Workweek, Experience, Quality of Practice, Development Limit and Expected Productivity sectors.
3.3.4.1 The Labor Sectors

The Gross Labor sector (Figure 3-18) models the quantity of labor used in a development phase. Headcount is assumed to be the number of full time, rested, experienced product developers. Therefore a rested experienced developer which spent half of his or her time developing the product would be considered 0.50 person. Headcount is changed within the range set by the minimum and maximum headcount to bring the actual headcount to the required headcount. The required headcount is based on the total work to be performed, the expected productivity of performing that work, and the time to the phase deadline. Gross Labor applied to the project phase is the product of the headcount and workweek is the gross labor. Fractions calculated in the Labor Allocation sector are used to allocate the gross labor among basework, rework, quality assurance, and coordination.

![Diagram of the Gross Labor Sector](image)

Figure 3-14: The Gross Labor Sector

The equations used to model the gross labor sector are described next.

$$\text{Gross Labor(Phase)} = \text{Headcount(Phase)} \times \text{Workweek(Phase)}$$

The gross labor available in a phase is the product of the headcount and the average workweek of the developers in the phase.

$$\text{Headcount(Phase)} = \text{Headcount(Phase)} + dt \times (\text{Change Headcount(Phase)})$$
Headcount(Phase) = Initial_Headcount(Phase)

Change_Headcount(Phase) = MIN(Max_Headcount(Phase) - Headcount(Phase), (Required_Headcount(Phase) - Headcount(Phase)) / Headcount_Adjustment_Time(Phase))

A phase's headcount moves from the headcount at which the phase begins toward the headcount required by the phase. A maximum headcount for the phase limits the rate of headcount growth. The movement of the current headcount is smoothed and slowed by a time which represents the delay in adding developers to or releasing developers from a project phase.

Required_Headcount(Phase) = FIFZE((Required_Personweeks(Phase) * Budget_Rqrd_Headct_effect) / (Time_to_Deadline(Phase)), 0, StoppedFlag(Phase))

Required_Personweeks(Phase) = Total_Labor_Required(Phase) / Normal_Workweek(Phase)

Budget_Rqrd_Headct_effect = TABHL(TC1, Budget_Status, -1.00, 0, 0.10)

$T_{TC1} = 0.00/0.05/0.10/0.20/0.25/0.35/0.40/0.50/0.70/0.90/1.00$

The required headcount is the number of developers required to complete the remaining work by the phase's deadline if all developers work a normal workweek (typically 40 hours per week). A project over budget will reduce this number in an attempt to control spending. This is based on the assumption that project managers will feel pressure to keep headcount and thereby costs down to bring project costs closer to the project budget.

Total_Labor_Required(Phase) = RemainingWork(Phase) / AvgPrody(Phase)

RemainingWork(Phase) = Task_List(Phase) - Tasks_Released(Phase)

AvgPrody(Phase) = (Ref_Coord_Prdctvty(Phase) + Ref_BW_Prdctvty(Phase) + Ref_RW_Prdctvty(Phase) + Ref_QA_Prdctvty(Phase)) / 4

The labor required to complete a phase is the sum of the unfinished tasks divided by the average productivity of the four development activities. The remaining work includes basework, rework and tasks waiting for quality control. The productivities are weighted equally on the assumption that developers are not likely to incorporate the amount of each kind of work remaining in their estimates of total labor required.
The Labor Allocation sector (Figure 3-15) calculates the fraction of the gross labor which developers apply to the basework, quality assurance, rework, and coordination activities. For each of these activities the work available from the process and the productivity expected by developers of each activity are used to determine a pressure for applying labor to each activity. These pressures are adjusted based on the priority of the activity and the performance relative to targets for schedule, quality, and cost. The labor fractions are the fraction of the total pressure for each activity.

![Labor Allocation Diagram]

**Figure 3-15: The Labor Allocation Sector**

The equations used to model the allocation of labor are described below.

\[
\text{Coord\_Labor\_Required(Phase)} = \frac{\text{Coord\_Process\_Limit(Phase)}}{\text{Expect\_Coord\_Prdctvty(Phase)}} \\
\text{BW\_Labor\_Required(Phase)} = \frac{\text{BW\_Process\_Limit(Phase)}}{\text{Expect\_BW\_Prdctvty(Phase)}} \\
\text{RW\_Labor\_Required(Phase)} = \frac{\text{RW\_Process\_Limit(Phase)}}{\text{Expect\_RW\_Prdctvty(Phase)}} \\
\text{QA\_Labor\_Required(Phase)} = \frac{\text{QA\_Process\_Limit(Phase)}}{\text{Expected\_QA\_Prdctvty(Phase)}}
\]
The labor required for each development activity is the number of person-hours required to complete the currently available work as determined by the process limit. This is the activity's process limit divided by the developer's expected productivity for that activity.

\[
\text{Press}_{\text{for BW}}(\text{Phase}) = \text{EXP}((\text{BW}_{\text{Labor Required}}(\text{Phase}) \times \text{BW}_{\text{Priority}}(\text{Phase}) \times \\
\text{Sched}_{\text{BW Press effect}}(\text{Phase}) \times \text{Cost effect on BW Import})/\alpha(\text{Phase}))
\]

\[
\text{Press}_{\text{for RW}}(\text{Phase}) = \text{Quality}_{\text{Goal Switch}}(\text{EXP}((\text{RW}_{\text{Labor Required}}(\text{Phase}) \times \\
\text{RW}_{\text{Priority}}(\text{Phase}) \times \text{Qual Gap effect on QARW priority})/\alpha(\text{Phase})))
\]

\[
\text{Press}_{\text{for Coord}}(\text{Phase}) = \text{Quality}_{\text{Goal Switch}}((\text{EXP}((\text{Coord}_{\text{Labor Required}}(\text{Phase}) \times \\
\text{Coord}_{\text{Priority}}(\text{Phase}) \times \text{Qual Gap effect on Coord Import})/\alpha(\text{Phase})))
\]

\[
\text{Press}_{\text{for QA}}(\text{Phase}) = \text{Quality}_{\text{Goal Switch}}((\text{EXP}((\text{QA}_{\text{Labor Required}}(\text{Phase}) \times \\
\text{QA}_{\text{Priority}}(\text{Phase}) \times \text{Qual Gap effect on QARW priority})/\alpha(\text{Phase})))
\]

The pressure felt by the developers to use their available time for each of the development activities increases exponentially with increases in the labor required and the priority given to the activity. Low quality performance relative to the quality standard increases the pressure for rework, quality assurance and coordination.

\[
\text{Qual Gap effect on QARW priority}(\text{Phase}) = \text{TABHL}(\text{TQ2,Current Quality}(\text{Phase}) - \\
\text{Quality}_{\text{Goal}}(\text{Phase}), -1.00, 0.00, 0.10)
\]

\[
\text{TQ2}=2.20/2.14/2.07/1.99/1.90/1.80/1.68/1.54/1.38/1.20/1.00
\]

\[
\text{Current Quality}(\text{Phase}) = 1-(\text{Known Rework}(\text{Phase})/(\text{Tasks Released}(\text{Phase})))
\]

The quality performance in each phase influences the priority given to quality assurance and rework in the allocation of labor. The current quality is the from the perspective of the developer and is the percent of tasks released which are believed to be free of errors. When the current quality is greater than the quality goal (described later) the priorities given to rework and quality assurance are not changed. But when current quality is worse than the quality goal the priority of rework and quality assurance increases to reflect the developer's attempts to correct for poor quality by putting more of the available time into the development activities which most directly influence the number of errors. An upper limit to this influence is assumed to exist and be a factor of 2.20.

\[
\text{Sched}_{\text{BW Press effect}}(\text{Phase}) = \text{TABHL}(\text{TL7,Sched Pressure}(\text{Phase}), 0.5, 0.50)
\]

\[
\text{T TL7}=1.00/1.01/1.04/1.07/1.12/1.17/1.25/1.32/1.44/1.58/1.73
\]
Poor schedule performance is modeled with schedule pressure (described later). This table can reflect a specific development organization's perspective of schedule impacts. In this table no level of schedule pressure decreases the priority of basework, reflecting a strong emphasis on schedule performance. As the schedule pressure increases the priority given to basework increases. This represents the developer's efforts to finish the project more quickly by focusing their available labor on the most direct and obvious reflection of progress. This representation reflects a possible simplification of the project in the mental models of the developers in that it assumes that at an operational level developers perceive increased basework as the only change in labor priority appropriate when a project gets behind schedule.

\[
\begin{align*}
\text{Labor\_Fraction\_to\_Coord(Phase)} &= \frac{\text{Press\_for\_Coord(Phase)}}{	ext{Total\_Pressure\_for\_Activities(Phase)}} \\
\text{Labor\_Fraction\_to\_BW(Phase)} &= \frac{\text{Press\_for\_BW(Phase)}}{	ext{Total\_Pressure\_for\_Activities(Phase)}} \\
\text{Labor\_Fraction\_to\_RW(Phase)} &= \frac{\text{Press\_for\_RW(Phase)}}{	ext{Total\_Pressure\_for\_Activities(Phase)}} \\
\text{Labor\_Fraction\_to\_QA(Phase)} &= \frac{\text{Press\_for\_QA(Phase)}}{	ext{Total\_Pressure\_for\_Activities(Phase)}} \\
\text{Total\_Pressure\_for\_Activities(Phase)} &= \text{Press\_for\_QA(Phase)} + \text{Press\_for\_Coord(Phase)} + \text{Press\_for\_BW(Phase)} + \text{Press\_for\_RW(Phase)}
\end{align*}
\]

The fraction (as a percentage) of the total available labor allocated to each of the development activities is the fraction of the total labor pressure felt by developers to perform the specific development activity. The total pressure for labor is the sum of the pressures for the four development activities.

\[
\begin{align*}
\text{BW\_Labor(Phase)} &= \text{BW\_Labor(Phase)} + \frac{\text{Labor\_Fraction\_to\_BW(Phase)\times dt\times Gross\_Labor(Phase)/BW\_Labor\_Delay(Phase)\times BW\_Labor(Phase)}}{BW\_Labor(Phase)} \\
\text{RW\_Labor(Phase)} &= \text{RW\_Labor(Phase)} + \frac{\text{Labor\_Fraction\_to\_RW(Phase)\times dt\times Gross\_Labor(Phase)/RW\_Labor\_Delay(Phase)\times RW\_Labor(Phase)}}{RW\_Labor(Phase)} \\
\text{Coord\_Labor(Phase)} &= \text{Coord\_Labor(Phase)} + \frac{\text{Labor\_Fraction\_to\_Coord(Phase)\times dt\times Gross\_Labor(Phase)/Coord\_Labor\_Delay(Phase)\times Coord\_Labor(Phase)}}{Coord\_Labor(Phase)} \\
\text{QA\_Labor(Phase)} &= \text{QA\_Labor(Phase)} + \frac{\text{Labor\_Fraction\_to\_QA(Phase)\times dt\times Gross\_Labor(Phase)/QA\_Labor\_Delay(Phase)\times QA\_Labor(Phase)}}{QA\_Labor(Phase)}
\end{align*}
\]
The labor allocated to each development activity is the smoothed product of the activity’s labor fraction and the gross labor available. The first order smooth represents the delay between the generation of the pressure for a development activity and the allocation of labor to that activity. This delay is an estimate of the effects of organizational inertia and the flexibility of developers to shift from one development activity to another.

Modeling the size of the average workweek and the effects of fatigue are done by the Workweek sector (Figure 3-16). The current workweek moves within a range (0 - 140 hours per week) relative to a normal workweek (40 hours per week) based on the impacts of schedule pressure. This value is averaged over the recent past to represent developer fatigue, which impacts the quality of work and productivity.

![Diagram of Workweek sector]

**Figure 3-16: The Workweek Sector**

The equations used to model the workweek sector are described next.

\[
\text{Avg}_\text{Wrkwk}(\text{Phase}) = \text{Avg}_\text{Wrkwk}(\text{Phase}) + dt*(\text{Avg}_\text{Wrkwk}_\text{Change}(\text{Phase}))
\]

\[
\text{Avg}_\text{Wrkwk}_\text{Change}(\text{Phase}) = (\text{Workweek}(\text{Phase}) \text{Avg}_\text{Wrkwk}(\text{Phase})) / \text{Wrkwk}_\text{Avg}_\text{Time}(\text{Phase})
\]

The average workweek in each phase moves from the initial normal workweek (40 hours per week) toward the current workweek at a rate slowed by the time required for working more or less than the average to have an impact on the developers. For example if the average workweek was 40 hours per week and the developers were relatively rested and the current workweek suddenly jumped to 60 hours per week the impacts would not be
immediately felt by the developers. But as the current workweek remained high fatigue would develop slowly. The slow growth in the average workweek reflects this change.

\[
\text{Workweek(Phase)} = \text{MIN(Normal\_Workweek(Phase)} \times \\
\text{Sched\_Workweek\_effect(Phase)}, \text{Max\_Workweek(Phase)})
\]

\[
\text{Sched\_Workweek\_effect(Phase)} = \text{TABHL(TL10,Time\_Required(Phase)/}
\text{Time\_to\_Deadline(Phase)},0,5,0.5)
\]
\[
T \text{TL10}=0.97/0.99/1.0/1.05/1.15/1.30/1.50/1.80/2.20/2.70/3.30
\]

The workweek is the developers response to the pressure to keep the project on schedule but remains within a maximum workweek size. There is very little response to being ahead of schedule, when the ratio of time required to the time available (the schedule pressure) is less than 1. This is consistent with the table reflecting the developer response to schedule pressure on basework priority. But as the schedule pressure increases the increase in workweek grows exponentially to a maximum multiplier of 3.30. This represents and extreme condition of developers feeling pressure to work 132 hours per week (at 40 hour per week normal workweek). While actually working this many hours is unreasonable, developers feeling the pressure to work this many hours in response to extremely high schedule pressure is possible and a reasonable upper bound.

3.3.4.2 The Experience Sector

The Experience sector (Figure 3-17) models the cumulative experience of the development team and its impact on productivity. Cumulative experience is measured in experience units which are the same size as development tasks. These are accumulated by the development team by performing basework and by increases in headcount, which represent the addition of more developers to the project phase. Accumulated experience decreases with decreases in headcount, representing developers leaving the project phase. The accumulated experience affects the productivity through an exponential learning curve relationship in which the productivity increases at a decreasing rate with more and more experience.
Figure 3-17: The Experience Sector

The equations used to model the experience sector are described next.

\[
\text{Cumm}_\text{Exper}(\text{Phase}) = \text{Cumm}_\text{Exper}(\text{Phase}) + \text{dt} \times (\text{Change}_\text{Cum}_\text{Exper}(\text{Phase}))
\]

\[
\text{Cumm}_\text{Exper}(\text{Phase}) \text{ [initial value]} = \text{Avg}_\text{New}_\text{member}_\text{Exper}(\text{Phase}) \times \text{Initial}_\text{Headcount}(\text{Phase})
\]

\[
\text{Change}_\text{Cum}_\text{Exper}(\text{Phase}) = \frac{\text{Net}_\text{Exper}_\text{Gain}(\text{Phase})}{\text{Exper}_\text{Assim}_\text{Time}(\text{Phase})}
\]

Experience is measured in units which each represent the lesson learned from performing a single development task. The cumulative experience of the developers working on each phase at the beginning of the phase is assumed to be the product of the starting headcount of the phase and the experience level of a new development team member. The (effective) cumulative experience of the team adjusts to actual changes in the team experience at a rate slowed by the time required to assimilate those changes into its development work. This time represents the time to translate experience into useful knowledge and the time required to start to apply that useful knowledge.

\[
\text{Net}_\text{Exper}_\text{Gain}(\text{Phase}) = \text{Basework}(\text{Phase}) + \text{New}_\text{Memb}_\text{Exper}_\text{Gain}(\text{Phase}) - \text{Exper}_\text{Lost}(\text{Phase})
\]
The actual experience of the team increases with each performance of a new development task. It is assumed that learning occurs primarily in the performance of initial development work and that no additional experience is gained by quality assurance, rework, or coordination. The actual experience of the team also increases with the additional of new team members and decreases with the loss of team members.

\[
\text{New\_Memb\_Exper\_Gain(Phase)} = \text{Avg\_New\_member\_Exper(Phase)} \times \text{MAX(0,Change\_Headcount(Phase))}
\]

\[
\text{Exper\_Lost(Phase)} = (-1) \times \text{MIN(0,Change\_Headcount(Phase))} \times \text{Avg\_memb\_Exper(Phase)}
\]

\[
\text{Avg\_memb\_Exper(Phase)} = \text{FIFGE(Cumm\_Exper(Phase)/Headcount(Phase),Cumm\_Exper(Phase),Headcount(Phase),1.0)}
\]

New team members add experience at a low "new member" level. Experience is lost at the rate of the average experience level achieved by the development team.

\[
\text{Exper\_index(Phase)} = (0.80)^{\text{LOGN(Cumm\_Exper(Phase)/Ref\_Exper(Phase))}/\text{LOGN(2)}}
\]

Team experience influences other project features through a "learning curve" effect. The learning curve effect is modeled by calculating an influence factor which shrinks 20% for every doubling of cumulative experience of the team. The model uses a reference experience level and the team's cumulative experience to determine the number of doublings. The Experience Index decreases as the development team gains experience.

3.3.4.3 The Quality of Practice Sector

The Quality of Practice sector (Figure 3-18) models the impacts of schedule (working faster), experience (working smarter), coordination (working more effectively with others), and fatigue (working tired) on the quality of work performed by the developers. A reference level of quality is adjusted up with more coordination and experience and down with more schedule pressure and fatigue. A phase's quality of practice impacts its error generation and discovery rates.
Figure 3-18: The Quality of Practice Sector

The equations used to model the quality of practice sector are described next.

\[
\text{Quality of Practice(Phase)} = \text{Ref Qual of Practice(Phase)*Exper effect on QofP(Phase)*Sched Qual of Prac effect(Phase)*Fatigue Qual of Prac effect(Phase)*Coord effect on QofP(Phase)}
\]

The quality of practice of the developers in each phase is based on a reference value which is altered by developer experience, schedule pressure, fatigue, and coordination. Each of the influences represents a different cause of changes in the quality of practice. More experience improves the quality of practice because developers are "working smarter".

\[
\text{Exper effect on QofP(Phase)} = \text{TABHL(T7,Exper index(Phase),0,5,0.50)}
\]
\[
T T7=2.50/2.4/2.2/1.9/1.5/1.00/0.8/0.6/0.5/4/35
\]

More experience improves the quality of practice. This relationship reflects that developers are "working smarter" as a team within a phase when they have more experience. Early in the project when the development team has relatively little experience and the index is large the lack of experience decreases the team's quality of practice. As the team gains experience by performing basework the index drops and the effect on the quality of practice improves to a maximum impact of increasing the quality of practice by a factor of 2.50.
Sched_Qual_of_Prac_effect(Phase)=TABHL(T11,Sched_Pressure(Phase),1,10,0.90)
T T11=1/0.99/0.97/0.94/0.90/0.85/0.79/0.72/0.54/0.55/0.45

Increasing schedule pressure decreases the quality of practice because the developers are "working faster" to recover the time and therefore not doing as good a job. Schedule pressure is assumed to only hurt and never help the team's quality of practice. A lower limit is placed on this relationship, reflecting the assumption that professional developers will retain some quality of practice even under extremely high schedule pressure conditions.

Fatigue_Qual_of_Prac_effect(Phase)=TABHL(TL12,Avg_Wrkwk(Phase)/
Normal_Workweek(Phase),0.5,0.50)
T TL12=1.05/1.05/1.0/0.98/0.94/0.88/0.80/0.70/0.58/0.44/0.44

More fatigue decreases the quality of practice because the developers are "working tired". Fatigue is modeled as the response to the ratio of the average workweek over a period of time (described above) to the normal workweek. The relationship of fatigue to the quality of practice is nonlinear with little influence when the average workweek is less than normal. This reflects the assumption that any time made available due to needing to work on the project less than the normal workweek will be absorbed (according to Parkinson's Law). A maximum effect is reached as the average workweek exceeds the normal by a factor of five.

Coord_effect_on_QofP(Phase)=TABHL(TL3,Coord_Status(Phase),0.2,0.20)
T TL3=0.00/0.06/0.18/0.36/0.6/0.9/1.28/1.66/1.84/1.96/2.00

More coordination increases the quality of practice because the developers are "working more effectively". This is because their interaction with other development phases has given them improved knowledge and insight into their part of the entire project. Coordination only improves the quality of practice in this relationship.

QA_Status(Phase)=QA_Labor(Phase)/(QA_Labor_Required(Phase))

The status of the quality of practice in each of the phases is the ratio of the labor provided for quality assurance to the labor required for quality assurance work.
3.3.4.4 The Expected Productivity Sectors

The four Expected Productivity sectors (Figures 3-19, 3-20, 3-21, and 3-22) model developer's perceptions of the productivity of development activities. These expectations are used to simulate the pressures felt by developers to apply their time to the different development activities. Each of the four perceived productivities (for basework, quality assurance, rework, and coordination) are based on the actual productivity experienced and altered by the delay in reporting and adjusting of productivity expectations.

![Diagram of Expected Basework Productivity](image)

**Figure 3-19: The Expected Basework Productivity Sector**

![Diagram of Expected QA Productivity](image)

**Figure 3-20: The Expected Quality Assurance Productivity Sector**
The equations used to model the expected productivity sectors are described next.

\[
\text{Expect\_BW\_Prdctvty(Phase)} = \text{Expect\_BW\_Prdctvty(Phase)} + dt \times (\text{Change\_Expect\_BW\_Prdctvty(Phase)})
\]

\[
\text{Expect\_RW\_Prdctvty(Phase)} = \text{Expect\_RW\_Prdctvty(Phase)} + dt \times (\text{Change\_Expect\_RW\_Prdctvty(Phase)})
\]
Expected_QA_Prdctvty(Phase)=Expected_QA_Prdctvty(Phase)+dt* (Change_Expect_QA_Prdctvty(Phase))

Expect_Coord_Prdctvty(Phase)=Expect_Coord_Prdctvty(Phase)+dt* (Change_Expect_Coord_Prdctvty(Phase))

The developers in each phase generate expectations about their productivity at performing the four development activities. These expected productivities move from initial reference values which are estimates of historical productivity expectations toward actual productivities.

Change_Expect_BW_Prdctvty(Phase)= (Actual_BW_Prdctvty(Phase)-
Expect_BW_Prdctvty(Phase))/Adjust_Expect_BW_Prdctvty_Time(Phase)

Change_Expect_RW_Prdctvty(Phase)= (Actual_RW_Prdctvty(Phase)-
Expect_RW_Prdctvty(Phase))/Adjust_Expect_RW_Prdctvty_Time(Phase)

Change_Expect_QA_Prdctvty(Phase)= (Actual_QA_Prdctvty(Phase)-
Expected_QA_Prdctvty(Phase))/Adjust_Expect_QA_Prdctvty_Time(Phase)

Change_Expect_Coord_Prdctvty(Phase)= (Actual_Coord_Prdctvty(Phase)-
Expect_Coord_Prdctvty(Phase))/Adjust_Expect_Coord_Prdctvty_time(Phase)

Changes in productivity expectations are slowed by a time which represents the time needed for developers to build the newly experienced productivity of the current project into their expectations about how they will perform in the future. This can require the alteration of long-held and personally important performance images which the developers hold.

BW_Labor_Limit(Phase)=Act_BW_Prd(Phase)*BW_Labor(Phase)

RW_Labor_Limit(Phase)=Act_RW_Prd(Phase)*RW_Labor(Phase)

QA_Labor_Limit(Phase)=Act_QA_Prd(Phase)*QA_Labor(Phase)

Coord_Labor_Limit(Phase)=Act_Coord_Prd(Phase)*Coord_Labor(Phase)

The limit placed on the development activity rate for each of the four development activities in each phase is the product of the actual productivity of that activity in the phase and the amount of labor allocated to the activity. This assumes that management will not add more developers to a phase than the current available workload and
perceived productivities indicate are needed. This assumption may be unrealistic in some circumstances. The model could be altered to accommodate overloading of personnel by adding a structure to model the increase in labor provided due to managerial decisions to overload a development phase. The actual productivities of each development activity in each phase are formulated similar to the expected productivites. They start at initial reference values which are estimates of historical productivities experienced by the developers. The productivities move toward the most current productivity at a slowed rate and influenced by experience and coordination (see previous description).

\[
\text{Current\_BW\_Prdctvty(Phase)} = \frac{\text{Basework(Phase)}}{\text{BW\_Labor(Phase)}}
\]

\[
\text{Current\_RW\_Prdctvty(Phase)} = \frac{\text{Rework(Phase)}}{\text{RW\_Labor(Phase)}}
\]

\[
\text{Current\_QA\_Prdctvty(Phase)} = \frac{\text{QA\_inspection\_rate(Phase)}}{\text{QA\_Labor(Phase)}}
\]

\[
\text{Current\_Coord\_Prdctvty(Phase)} = \frac{\text{Coord\_Limit(Phase)}}{\text{Coord\_Labor(Phase)}}
\]

The current productivity of each development activity in each phase is the rate at which that activity is being performed divided by the amount of labor applied to the activity. If the activity rate is being limited by the development process structure the current productivity can be much lower that if the labor structures limit activity.

\[
\text{Change\_Actual\_BW\_Prdctvty(Phase)} = \frac{\text{(Current\_BW\_Prdctvty(Phase))/}}{\text{Avg\_Act\_BW\_Prdctvty\_Time(Phase)})*\text{Exper\_on\_Prdctvty\_effect(Phase)}*}\text{Coord\_effect\_on\_Prdvy(Phase)}
\]

\[
\text{Change\_Actual\_RW\_Prdctvty(Phase)} = \frac{\text{(Current\_RW\_Prdctvty(Phase))/}}{\text{Avg\_Act\_RW\_Prdctvty\_Time(Phase)})*\text{Exper\_on\_Prdctvty\_effect(Phase)}*}\text{Coord\_effect\_on\_Prdvy(Phase)}
\]

\[
\text{Change\_Actual\_QA\_Prdctvty(Phase)} = \frac{\text{(Current\_QA\_Prdctvty(Phase))/}}{\text{Avg\_Act\_QA\_Prdctvty\_Time(Phase)})*\text{Exper\_on\_Prdctvty\_effect(Phase)}*}\text{Coord\_effect\_on\_Prdvy(Phase)}
\]

\[
\text{Change\_Actual\_Coord\_Prdctvty(Phase)} = \frac{\text{(Current\_Coord\_Prdctvty(Phase))/}}{\text{Avg\_Act\_Coord\_Prdctvty\_Time(Phase)})*\text{Exper\_on\_Prdctvty\_effect(Phase)}}
\]

Actual productivities move at a smoothed rate which represents the time required for instantaneous productivities to become effective (no smooth causes unrealistic large fluctuations) as effected by the level of team experience and coordination.
Exper_on_Prdctvty_effect(Phase)=TABHL(TL13,Exper_index(Phase),0,5,0.50)
T TL13=1.33/1.30/1.24/1.18/1.1/1.00/0.92/0.82/0.76/0.72/0.70

More experience increases productivity. This relationship reflects that developers are "working smarter" as a team within a phase when they have more experience. The impact described by this relationship remains within a relatively narrow range (33% increase to 30% decrease in productivity). The lower limit reflects that productivity levels have minimums based on levels of training and skill which little experience cannot erode. The upper limit reflects a maximum impact of experience.

Coord_effect_on_Prdy(Phase)=TABHL(TL2,Coord_Status(Phase),0,1,0.10)
T TL2=0.195/0.41/0.575/0.725/0.825/0.89/0.945/0.96/0.975/0.985/1.00

The level of coordination also impacts productivity. Inadequate coordination decreases productivity because developers who do not communicate and coordinate their work are "working less effectively". This is because interaction with other development phases can give them improved knowledge and insight into their part of the entire project. Excess coordination is assumed to not impact productivity. Inadequate coordination has a limited reduction in productivity, reflecting that even developers who do not coordinate their work with other phases at all are able to proceed with development activities.

3.3.5 The Targets and Performance Subsystems

Like the Resources subsystems, the structural components of the Targets and Performance subsystems are based upon existing system dynamics models which are referenced in Table 3-1. The targets and performance subsystems model project goals and actual performance in three dimensions: time, quality and money. These subsystems consist of the Schedule, Quality, and Cost sectors.

3.3.5.1 The Schedule Sector

Schedule goals and performance are modeled at both the project and phase level. At the project level the time objective is set with the project deadline. Project performance in the
time domain is measured by the actual project completion time. The Actual completion time is compared to the project deadline to evaluate project schedule performance.

The project deadline moves toward the projected completion date when the schedule pressure exceeds the capacity of the team to resist that pressure (Figure 3-23). Resistance to schedule slippage slows this movement. Schedule pressure is defined as the ratio of the time to the expected completion date to the time to the current project deadline. The expected completion date for the project is the expected completion date of the last development phase.

![Figure 3-23: The Project Deadline Sector](image)

A similar structure is used to model the schedule in each of the development phases. Expected phase completion dates are based on the remaining labor required and the available labor. Schedule pressure in each phase is defined the same as at the project level and impacts the allocation of labor within the phase, quality of practice, error discovery, and length of workweek.
The equations used to model the project and phase schedules are described next.

\[ \text{Project\_Deadline} = \text{Project\_Deadline} + \text{dt} \times (\text{Proj\_Sched\_Slip}) \]

\[ \text{Proj\_Sched\_Slip} = \text{FIFGE} (\text{Expected\_Proj\_Completion\_Time} - \text{Project\_Deadline}, 0, \text{Proj\_Sched\_Press}, \text{Resistance\_to\_Sched\_Slip}) \]

The project deadline moves from its current value to the expected completion time when the schedule pressure for the entire project exceeds the threshold for schedule pressure acceptable.

\[ \text{Proj\_Sched\_Press} = \text{MAX} ((\text{Expected\_Proj\_Completion\_Time} - \text{TIME}) / (\text{Project\_Deadline} - \text{TIME}), (\text{Project\_Deadline} - \text{TIME}) / (\text{Expected\_Proj\_Completion\_Time} - \text{TIME})) \]

The schedule pressure is defined as the ratio of the time required to complete the project and the time available until the current project deadline. The maximum of the inverse ratios is used to capture schedule pressure due to expected completions before and after the current deadline.

\[ \text{Expected\_Proj\_Completion\_Time} = \text{MAX} (\text{ExpCompl\_Time}(1), \text{ExpCompl\_Time}(2), \text{ExpCompl\_Time}(3), \text{ExpCompl\_Time}(4), \text{ExpCompl\_Time}(5)) \]

The expected project completion time is the latest expected completion time of the development phases.

Project schedules of each of the development phases is modeled in the same manner as for the project as a whole.

\[ \text{DL(Phase)} = \text{DL(Phase)} + \text{dt} \times \text{Change\_DL(Phase)} \]

\[ \text{Change\_DL(Phase)} = \text{FIFGE} (\text{ExpCompl\_Time(Phase)} - \text{DL(Phase)}, 0, \text{Sched\_Pressure(Phase)}, \text{Resistance\_to\_Sched\_Slip}) \]

The deadline for each phase moves from its current value to the expected completion time when the schedule pressure for the phase exceeds the threshold for schedule pressure acceptable.

\[ \text{Sched\_Pressure(Phase)} = \text{MAX} ((\text{ExpCompl\_Time(Phase)} - \text{TIME}) / (\text{DL(Phase)} - \text{TIME}), (\text{DL(Phase)} - \text{TIME}) / (\text{ExpCompl\_Time(Phase)} - \text{TIME})) \]
The schedule pressure is defined as the ratio of the time required to complete the phase and the time available until the current phase deadline. The maximum of the inverse ratios is used to capture schedule pressures due to expected completions before and after the current deadline.

\[
\text{ExpComplTime(Phase)} = \text{ExpStartTime(Phase)} + \text{ExpDur(Phase)}
\]

The expected completion time of each phase is the time the phase is expected to start plus the expected duration of the phase.

\[
\text{ExpStartTime(2)} = (1-\text{StartFlag(2)}) \times ((\text{Task\_List(1)/TotalTaskList}) \times \text{Project\_Deadline}) + \text{StartFlag(2)} \times \text{Start\_Time(2)}
\]

The time of the expected start time of each phase from the project start is estimated to be proportional to the size of phase to the project size if the phase has not started. This estimate is required because the exact starting time of phases are not known until the phase actually begins. The time of the expected start time of each phase from the project start for active or completed phases is the phase's actual starting time. This formulation must estimate the concurrence of dependent development phases. It cannot identify the longest dependent path of phases through the project and add expected durations along that path because that assumes totally sequential phases, which is a very false assumption and greatly exaggerates the expected completion time. The "StartFlag" parameter acts a switch between the estimate of future phases and the documentation of the start time of active or completed phases. Phase 2 is shown as an example of this formulation. Software limitations prevent the generalization of this formulation.

\[
\text{ExpDur(Phase)} = \text{Time\_spent\_to\_Date(Phase)} + \text{Time\_Required(Phase)}
\]

\[
\text{Time\_Required(Phase)} = \text{RemainingWork(Phase)} / (\text{AvgPrody(Phase)} * \text{Normal\_Workweek(Phase)} * (\text{Max\_Headcount(Phase)} / 2))
\]

\[
\text{AvgPrody(Phase)} = (\text{Ref\_Coord\_Prdctvty(Phase)} + \text{Ref\_BW\_Prdctvty(Phase)} + \text{Ref\_RW\_Prdctvty(Phase)} + \text{Ref\_QA\_Prdctvty(Phase)}) / 4
\]

\[
\text{RemainingWork(Phase)} = \text{MAX}(0, (0.999 \times \text{Task\_List(Phase)}) - \text{Tasks\_Released(Phase)})
\]

The expected duration of a phase is the time spent so far on the phase plus the estimate of the time required to complete the phase. The time spent to date is the time elapsed from
the start of the phase. The estimate of the time required to complete the phase is the time necessary to complete the remaining work if the average historical (reference) productivities are applied by the average labor force. The average labor force is assumed to be the half the maximum headcount working the normal workweek. This formulation must differ from many in the model which reflect the developer and manager's perception of primarily the currently available development work. Developers and managers acknowledge the total remaining visible work in estimating required durations and see beyond the currently available work. Therefore this estimate must utilize the total work remaining to be done in the phase. The formulation assumes optimistic developers in that the forecast assumes that none of the unstarted basework will require rework.

3.3.5.2 The Project Quality Sector

The Project Quality sector (Figure 3-24) models the movement of the project quality goal from its initial level toward the current known quality level. Quality is measured in the percent of tasks which are known by the developers to be flawed. The time to adjust the quality goal slows the migration of the goal. Project quality below the project goal increases pressure for quality assurance, rework, and coordination work.

![Project Quality Diagram](image)

**Figure 3-24: The Project Quality Sector**

The equations used to model the project quality sector are described next.

\[
\text{Quality\_Goal} (\text{Phase}) = \text{Quality\_Goal} (\text{Phase}) + dt \times (\text{Change\_Quality\_Goal} (\text{Phase}))
\]
Change_Quality_Goal(Phase) = (Current_Quality(Phase) - Quality_Goal(Phase))/Quality_Goal_Adjust_Time(Phase)

The quality goal of each phase moves from its initial value toward the current quality of that phase at a rate determined by the time required to adjust that goal. These movements are relatively small since the difference between the goal and actual quality is often relatively small and the adjustment time is relatively long.

Current_Quality(Phase) = 1 - (Known_Rework(Phase))/(Tasks_Released(Phase) + Tasks_Completed_not_Checked(Phase)+Known_Rework(Phase))

The current quality of each phase is the percent of tasks completed once which are believed to not be flawed. This reflects the developer's perspective (optimism) that unchecked tasks are not flawed. This formulation uses the complete project work instead of the release of tasks by the final project phase because during the project the latter information is not available to developers or managers.


Change_Proj_Quality_Goal = (Current_Proj_Quality - Proj_Quality_Goal)/Proj_Quality_Goal_Adjust_Time

The quality goal of the entire project also moves from its initial value toward the current quality of the project at a rate determined by the time required to adjust that goal. These movements are relatively small since the difference between the goal and actual quality is often relatively small and the adjustment time is relatively long.

Current_Proj_Quality = 1 - (SUM(Known_Rework))/SUM(Tasks_Released)+SUM(Tasks_Completed_not_Checked)+SUM(Known_Rework))

The current quality of the project is the percent of tasks completed once which are believed to not be flawed. This reflects the developers perspective (optimism) that unchecked tasks are not flawed. This formulation uses the complete project work instead of the release of tasks by the final project phase because during the project the latter information is not available to developers or managers.

Proj_Quality_Gap = Current_Proj_Quality - Proj_Quality_Goal
The project's quality gap is the difference between the project quality goal and the current project quality.

\[
\text{Qual\_Gap\_effect\_on\_Coord\_Import} = \text{TABHL(TQ1, Proj\_Quality\_Gap, -1.00, 0.00, 0.10)} \\
T \ TQ1 = 2.10/1.90/1.72/1.56/1.42/1.30/1.20/1.12/1.06/1.02/1.00
\]

The project quality status impacts the priority given by developers to the allocation of available labor to the coordination activity. Quality that exceeds the quality goal has no impact on coordination priority. But poor quality performance increases the priority of coordination in an "S" shaped curve with an upper limit of a multiplier of 2.10.

3.3.5.3 The Cost Sector

The Cost sector accumulates project costs, projects total project costs, compares them to the project budget, and influences several parameters depending on the difference. The equations used to model costs are described next.

\[
\text{Project\_Cost\_to\_Date} = \text{SUM(Phase\_Cost\_to\_Date)}
\]

\[
\text{Phase\_Cost\_to\_Date(Phase)} = \text{Phase\_Cost\_to\_Date(Phase) + dt*Straight\_Cost(Phase)}
\]

The current cumulative cost of the project is the sum of the current cumulative costs of the project's phases. Each project phase cost rises with the addition of straight (salaried, no overtime premium) labor cost. Only straight time cost is used because the developers are salaried and are not compensated for overtime.

\[
\text{Straight\_Cost(Phase)} = \text{Avg\_Straight\_Pay(Phase)*Straight\_Time(Phase)*Cost\_Markup(Phase)}
\]

The incremental cost of straight labor is the product of the average hourly pay rate in dollars per hour, the time expended in hours, and the cost markup factor which represents the cost of overhead, fringe benefits, administrative support, employer-paid taxes, equipment, etc.

\[
\text{Straight\_Time(Phase)} = \text{Headcount(Phase)*Normal\_Workweek(Phase)}
\]
The amount of straight time is the number of developers currently working on the development phase times the hours per week used as the basis for payment (40 hours per week).

\[
\text{Tot\_Exp\_Costs} = \text{Forecasted\_Costs} + \text{Project\_Cost\_to\_Date}
\]

The total expected project cost is the sum of the cost of the project so far and the forecast of the remaining cost.

\[
\text{Forecasted\_Costs} = \text{Avg\_Cost} \times (\text{Project\_Deadline\_TIME})
\]
\[
\text{Avg\_Cost} = \text{Project\_Cost\_to\_Date}/(\text{TIME})
\]

The forecasted cost is the average cost of the project to date extended to the current project deadline. This tends to increase the expected project cost as the project deadline slips.

\[
\text{Budget\_Surplus} = \text{Proj\_Budget} - \text{Tot\_Exp\_Costs}
\]
\[
\text{Budget\_Status} = \text{Budget\_Surplus}/\text{Proj\_Budget}
\]

The budget surplus or deficit is the difference between the project's budget and the total expected cost of the project. The cost status of the project is represented in a form commonly used in practice, as the percent over (positive budget status) or under (negative budget status) budget.

Budget overruns impact other project factors. But being under budget has no impact. This is based on the assumption that developers do not believe that the underrun will exist for the duration of the project or that developers will use available funds by the end of the project. In either case, the underrun do not significantly influence the project.

\[
\text{Cost\_effect\_on\_BW\_Import} = \text{TABHL(TC2,Budget\_Status,-1.0,0.0,0.0,1.0)}
\]
\[
\text{TC2} = 1.87/1.58/1.35/1.17/1.06/1.00/0.98/0.95/0.89/0.81/0.65
\]

Cost performance influences the priority of basework activity. This is based on the assumption that developers will try to finish a project more quickly to constrain costs.
Therefore this relationship increases the priority of basework when projected costs exceed the budget.

### 3.3.6 Basis for Key Model Structures

The important model structures are based on previous system dynamics models, other project models and data based on field observations (described in more detail in Chapter 5). These are summarized in Table 3-1 below:

<table>
<thead>
<tr>
<th>Model Structure</th>
<th>Model References</th>
<th>Data based on Field Observations (Yes/No)</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Process Structure and Scope Subsystems</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Development Tasks sector</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Demand-driven process limits</td>
<td>none</td>
<td>Yes</td>
</tr>
<tr>
<td>Exponential smooth of demand for activities</td>
<td>Hannon and Ruth, 1994</td>
<td>No</td>
</tr>
<tr>
<td>Internal Available-work constraint</td>
<td>Homer et al., 1993</td>
<td>Yes</td>
</tr>
<tr>
<td>External Available-work constraint</td>
<td>Homer et al., 1993</td>
<td>Yes</td>
</tr>
<tr>
<td>Closed loop flow of tasks</td>
<td>Cooper, 1980, Richardson and Pugh, 1981, Ford et al., 1993</td>
<td>Yes</td>
</tr>
<tr>
<td><strong>Internal Errors sector</strong></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table 3-1: Basis for Important Model Structure Components (partial)
<table>
<thead>
<tr>
<th>Model Structure</th>
<th>Model References</th>
<th>Data based on Field Observations (Yes/No)</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Resources Subsystem</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td><em>Gross Labor sector</em></td>
<td>Perceived Resource needs</td>
<td>Yes</td>
</tr>
<tr>
<td></td>
<td>Jessen, 1988, 1990</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Abdel-Hamid, 1984</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Richardson and Pugh, 1981</td>
<td></td>
</tr>
<tr>
<td><em>Labor Allocation sector</em></td>
<td>Priority weighted pressure</td>
<td>Yes</td>
</tr>
<tr>
<td></td>
<td>Abdel-Hamid, 1984</td>
<td></td>
</tr>
<tr>
<td><em>Workweek sector</em></td>
<td>Average and side impacts</td>
<td>Yes</td>
</tr>
<tr>
<td></td>
<td>Kim, 1988</td>
<td></td>
</tr>
<tr>
<td><em>Fatigue effects</em></td>
<td>Homer, 1985</td>
<td>Yes</td>
</tr>
<tr>
<td></td>
<td>Abdel-Hamid, 1984</td>
<td></td>
</tr>
<tr>
<td><em>Experience sector</em></td>
<td>Learning curve/productivity effect</td>
<td>Yes</td>
</tr>
<tr>
<td></td>
<td>Abdel-Hamid and Madnick, 1991</td>
<td></td>
</tr>
<tr>
<td><em>Limits and Productivity sectors</em></td>
<td>Actual vs. perceived productivity</td>
<td>Yes</td>
</tr>
<tr>
<td></td>
<td>Richardson and Pugh, 1981</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Jessen, 1988</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Abdel-Hamid, 1984</td>
<td></td>
</tr>
<tr>
<td><strong>Targets and Performance Subsystems</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td><em>Schedule sector</em></td>
<td>Schedule pressure</td>
<td>Yes</td>
</tr>
<tr>
<td></td>
<td>Roberts, 1974</td>
<td></td>
</tr>
<tr>
<td><em>Perceived vs. actual progress</em></td>
<td></td>
<td>Yes</td>
</tr>
<tr>
<td></td>
<td>Roberts, 1974</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Richardson and Pugh, 1981</td>
<td></td>
</tr>
<tr>
<td><em>Schedule estimates</em></td>
<td>Abdel-Hamid, 1984</td>
<td>Yes</td>
</tr>
<tr>
<td><em>Deadline slippage</em></td>
<td>Richardson and Pugh, 1981</td>
<td>Yes</td>
</tr>
<tr>
<td><em>Project Quality sector</em></td>
<td>Quality goal slippage</td>
<td>No</td>
</tr>
<tr>
<td></td>
<td>Fiddaman, Oliva, and Aranda, 1993</td>
<td></td>
</tr>
<tr>
<td><em>Project Cost sector</em></td>
<td>Cost estimates</td>
<td>Yes</td>
</tr>
<tr>
<td></td>
<td>Abdel-Hamid and Madnick, 1991</td>
<td></td>
</tr>
</tbody>
</table>

Table 3-1: Basis for Important Model Structure Components (continued)
3.4 Model Behavior

3.4.1 Introduction

Plots of simulation results over time can help develop an understanding of the model behavior. A few such results for two model configurations are provided to develop a fundamental understanding of the model. Exact parameter values and simulation output are not as important as an intuitive understanding of the impacts of the model's structure on its behavior. Simulation under specific sets of parameter values are addressed in the sensitivity analysis section and Chapter 5.

3.4.2 Typical Single Phase Model Behavior

Simulations of model behavior with a single phase and only the product development process and scope subsystems engaged provides a fundamental understanding of the system. Two plots will be shown. Figure 3-25 and 3-26 show the development tasks sector flows and stocks of this model configuration with no errors. The maximum basework rate set by the availability of tasks and the minimum duration is evident in Figure 3-25, as is the delay and smoothing impact of the structure on the Release Tasks flow. The resulting increase and then decline in the Completed not Checked stock and growth in the Tasks Released stock is seen in Figure 3-26.
Some of the impacts of errors can be seen in Figures 3-27 and 3-28 which show the flows and stocks of the development tasks sector with a 50% error generation rate. The flow of tasks due to the discovery of errors are shown in Figure 3-27. This flow responds to the basework rate in the initial peak and stable flows during the middle of the phase duration.
The influence of the delays and minimum durations in the system are shown in the variations in the flows at the end of the phase. The stocks which result from the integration of these flows are shown in Figure 3-28. The longer cycle time with errors (46 versus 24) is also evident by comparing Figures 3-25 and 3-26 to Figures 3-27 and 3-28.

Figure 3-27: Single Phase Simulation with 50% Error Generation Development Task Sector Flows

Figure 3-28: Single Phase Simulation with 50% Error Generation Development Task Sector Stocks
3.4.3 Typical Multiple Phase Model Behavior

A two phase configuration can show some of the important interactions between phases. Figures 3-29 and 3-30 show the flows and stocks for the upstream phase (labeled number 1) of a two phase configuration. Figures 3-31 and 3-32 show the flows and stocks for the downstream phase (labeled number 2) of a two phase configuration. Figure 3-29 shows three of the fundamental flows in the development tasks sector or the upstream phase. The delays seen in the single phase configuration are also evident here. Figure 3-30 also shows that the interaction between the two phases generates a small coordination backlog due to the discovery of errors by the downstream phase which have been released by the upstream phase.

![Graph showing development tasks per time unit for Basework, Rework, and Release Tasks.

Figure 3-29: Upstream Phase (1) Simulation of Two Phase Model Development Task Sector Flows]
Three of the fundamental development tasks sector flows of the downstream phase are shown in Figure 3-31. The delay caused by the external precedence relationship between the upstream and downstream phases is evident in the start of basework at time 12. The internal characteristics of the two phases are very similar. Therefore the smoothing effect of the upstream phase on the downstream phase can also be seen in Figure 3-31. The resulting lower stock values in the downstream phase are shown in Figure 3-32.
Figure 3-31: Downstream Phase (2) Simulation of Two Phase Model Development Task Sector Flows

Figure 3-32: Downstream Phase (2) Simulation of Two Phase Model Development Task Sector Stocks

The simulations above show some of the complexity which can be described and simulated with the Product Development Project Model. An understanding of the most influential portions of the model on behavior can be gained from a sensitivity analysis of the model.
3.5 Model Sensitivity to Parameter Values

3.5.1 Introduction

An important part of understanding model behavior is the identification of parameters to which model behavior is sensitive. These parameters can be the focus of parameter estimation work for model calibration and policy and system design. Model sensitivity to parameters is addressed in this section. Model parameter estimation for a specific set of conditions is discussed in Chapter 5: The Python Development Project. The model's sensitivity was tested at the single phase and multiple phase levels.

3.5.2 Parameter Reduction

Sensitivity analysis of the model requires the investigation of variables which are described with a single numerical value at any time (e.g. Minimum Activity Durations) and more complex variables such as Internal Precedence Relationships (Tank-Nielsen, 1980). The number of variables for testing was reduced by eliminating those which do not describe the operation of the real system (i.e. those used for managing the model). Additional reduction was possible by taking advantage of the fact that many parameters are described and used in sets of similar parameters. For example, the time to average actual productivity is used as the basis for a set of four parameters (the four development activities). In these cases the one parameter to which the system is considered most sensitive was kept and the others in the set eliminated for initial sensitivity testing. If initial testing revealed that a parameter set had high leverage the other parameters in the set were tested. The individual members of some sets of parameters was considered too important to apply this reduction (e.g. Minimum Activity Durations for the four development activities). Parameters were also eliminated if another parameter could be used to test the sensitivity of its feedback loop (e.g. delays in reported and expected productivities). Finally, parameters were eliminated if their value was strongly linked to the value of a selected parameter and could be tested through that selected parameter. For example high levels of experience in new team members can be expected to be associated with higher ability to resist schedule pressure. Twenty one parameters remained in a single phase model after these reductions.
3.5.3 Single Phase Model Sensitivity

Sensitivity was tested by observing project performance across a range of parameter values for a hypothetical but internally consistent test development project phase. The test phase is loosely based on the design activity of the project described in Chapter 5. Values for eliminated parameters were established at typical values, defined as a value estimated to be near the mean. Three sets of values were set for the twenty one selected parameters. Each set of values represents a consistent set of conditions. The first set of parameter values represents a pessimistic scenario. A likely scenario estimates the values of a typical project. The third set of values represents an optimistic scenario. The following values were assigned for these parameters for the pessimistic, likely, and optimistic scenarios.

<table>
<thead>
<tr>
<th>Parameter Name</th>
<th>Pessimistic Scenario Value</th>
<th>Likely Scenario Value</th>
<th>Optimistic Scenario Value</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Process Structure and Scope:</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>BW_Min_Task_duration</td>
<td>4</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>QA_Min_Task_Duration</td>
<td>3</td>
<td>1</td>
<td>0.5</td>
</tr>
<tr>
<td>RW_Min_Task_Duration</td>
<td>2</td>
<td>1</td>
<td>0.5</td>
</tr>
<tr>
<td>Task_List</td>
<td>1500</td>
<td>1000</td>
<td>500</td>
</tr>
<tr>
<td>Internal Precedence Relationship</td>
<td>linear</td>
<td>hyperbolic</td>
<td>open</td>
</tr>
<tr>
<td>Basic_prob_flawed_Task</td>
<td>0.9</td>
<td>0.5</td>
<td>0.3</td>
</tr>
<tr>
<td><strong>Resources:</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>BW_Priority</td>
<td>6</td>
<td>3</td>
<td>1.5</td>
</tr>
<tr>
<td>QA_Priority</td>
<td>0.20</td>
<td>1</td>
<td>5</td>
</tr>
<tr>
<td>RW_Priority</td>
<td>0.20</td>
<td>1</td>
<td>5</td>
</tr>
<tr>
<td>BW_Labor_Delay</td>
<td>6</td>
<td>2</td>
<td>0.5</td>
</tr>
<tr>
<td>QA_Labor_Delay</td>
<td>16</td>
<td>8</td>
<td>2</td>
</tr>
<tr>
<td>RW_Labor_Delay</td>
<td>4</td>
<td>1</td>
<td>0.25</td>
</tr>
<tr>
<td>Max_Headcount</td>
<td>1</td>
<td>3</td>
<td>5</td>
</tr>
<tr>
<td>Headcount_Adjustment_Time</td>
<td>16</td>
<td>8</td>
<td>4</td>
</tr>
<tr>
<td>Initial_Headcount</td>
<td>0.1</td>
<td>0.5</td>
<td>1.0</td>
</tr>
<tr>
<td>Exper_Assim_Time</td>
<td>4</td>
<td>1</td>
<td>0.5</td>
</tr>
<tr>
<td><strong>Targets and Performance:</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Initial_Proj_Deadline</td>
<td>25</td>
<td>50</td>
<td>100</td>
</tr>
<tr>
<td>Project_Quality_Goal</td>
<td>1.0</td>
<td>0.9</td>
<td>0.7</td>
</tr>
<tr>
<td>Project_Budget (x 1000)</td>
<td>75</td>
<td>125</td>
<td>250</td>
</tr>
<tr>
<td>Quality_Goal_Adjust_Time</td>
<td>48</td>
<td>12</td>
<td>4</td>
</tr>
<tr>
<td>Resistance_to_Sched_Slip</td>
<td>0.5</td>
<td>2</td>
<td>5</td>
</tr>
</tbody>
</table>

Table 3-2: Single Phase Sensitivity Test Parameter Values
Sensitivity is measured in the changes in project performance due to changes in parameter values. The three measures of project performance are cycle time, quality and cost. Cycle time is the time required for effectively all tasks (99.99%) to be released. Quality is measured by the number of flawed tasks released. Cost is the total expenses at completion. The test phase performance for the likely scenario is: 55 week cycle time, 363 defects released and $185,000 cost. Model sensitivity is the percent loss or improvement of project performance compared to the performance of the likely scenario due to changing a single parameter's value from the likely scenario value. The raw results of these tests are shown in Table 3-3 below. As an example, when the Quality Assurance Minimum Task Duration parameter is increased from 1 week (likely scenario) to 3 weeks (pessimistic scenario) the cycle time increases from 55 weeks (likely scenario) to 84 weeks (pessimistic scenario). This value is shown in the second column and third row of Table 3-3.

<table>
<thead>
<tr>
<th>Parameter Name</th>
<th>Project Cycle Time Performance (weeks)</th>
<th>Project Quality Performance (Defects Released)</th>
<th>Project Cost Performance ($ x 1000)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Likely Scenario Performance</td>
<td>55</td>
<td>363</td>
<td>185</td>
</tr>
<tr>
<td><strong>Process Structure and Scope:</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>BW_Min_Task_duration</td>
<td>67/55</td>
<td>363/398</td>
<td>223/165</td>
</tr>
<tr>
<td>QA_Min_Task_Duration</td>
<td>84/47</td>
<td>225/443</td>
<td>238/149</td>
</tr>
<tr>
<td>RW_Min_Task_Duration</td>
<td>70/48</td>
<td>364/365</td>
<td>213/168</td>
</tr>
<tr>
<td>Task_List</td>
<td>57/49</td>
<td>550/172</td>
<td>194/175</td>
</tr>
<tr>
<td>Internal Precedence Relationship</td>
<td>91/58</td>
<td>230/400</td>
<td>233/150</td>
</tr>
<tr>
<td>Basic_prob_flawed_Task</td>
<td>98/45</td>
<td>787/215</td>
<td>253/151</td>
</tr>
<tr>
<td><strong>Resources:</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>BW_Priority</td>
<td>55/55</td>
<td>364/363</td>
<td>185/185</td>
</tr>
<tr>
<td>QA_Priority</td>
<td>59/56</td>
<td>376/273</td>
<td>176/192</td>
</tr>
<tr>
<td>RW_Priority</td>
<td>55/54</td>
<td>363/387</td>
<td>185/182</td>
</tr>
<tr>
<td>BW_Labor_Delay</td>
<td>59/57</td>
<td>303/403</td>
<td>192/166</td>
</tr>
<tr>
<td>QA_Labor_Delay</td>
<td>59/49</td>
<td>354/193</td>
<td>223/204</td>
</tr>
<tr>
<td>RW_Labor_Delay</td>
<td>55/55</td>
<td>363/363</td>
<td>185/185</td>
</tr>
<tr>
<td>Max_Headcount</td>
<td>54/55</td>
<td>367/362</td>
<td>95/243</td>
</tr>
<tr>
<td>Headcount_Adjustment_Time</td>
<td>55/55</td>
<td>357/369</td>
<td>231/162</td>
</tr>
<tr>
<td>Initial_Headcount</td>
<td>72/54</td>
<td>390/372</td>
<td>129/190</td>
</tr>
<tr>
<td>Exper_Assim_Time</td>
<td>55/55</td>
<td>364/364</td>
<td>186/185</td>
</tr>
</tbody>
</table>

**Table 3-3: Single Phase Sensitivity Test Results**

Project Performance under Pessimistic and Optimistic Scenarios (partial)
Chapter 3

<table>
<thead>
<tr>
<th>Parameter Name</th>
<th>Project Cycle Time Performance (weeks)</th>
<th>Project Quality Performance (Defects Released)</th>
<th>Project Cost Performance ($ x 1000)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Initial Proj Deadline</td>
<td>51/60</td>
<td>339/398</td>
<td>196/161</td>
</tr>
<tr>
<td>Project Quality Goal</td>
<td>55/55</td>
<td>364/363</td>
<td>185/185</td>
</tr>
<tr>
<td>Project Budget (X1000)</td>
<td>55/55</td>
<td>358/363</td>
<td>150/203</td>
</tr>
<tr>
<td>Quality Goal Adjust Time</td>
<td>55/55</td>
<td>363/363</td>
<td>185/185</td>
</tr>
<tr>
<td>Resistance to Sched Slip</td>
<td>57/60</td>
<td>366/394</td>
<td>181/159</td>
</tr>
</tbody>
</table>

**Table 3-3: Single Phase Sensitivity Test Results**

*Project Performance under Pessimistic and Optimistic Scenarios (continued)*

The normalized results of these tests are shown in Table 3-4. For example the 29 week loss of schedule performance (84-55=29 weeks) in the previous example represents a 53% increase in cycle time (decrease in performance), as indicated by "-53" in Table 3-3. In a similar manner the "+15" represents an increase in project performance (decrease in cycle time) when the Quality Assurance Minimum Task Duration parameter is decreased from 1 week (likely scenario) to 0.5 week (optimistic scenario).

<table>
<thead>
<tr>
<th>Parameter Name</th>
<th>Project Cycle Time Performance Change (%)</th>
<th>Project Quality Performance Change (%)</th>
<th>Project Cost Performance Change (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Process Structure and Scope:</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>BW_Min_Task_duration</td>
<td>-22/0</td>
<td>0/-10</td>
<td>-21/+11</td>
</tr>
<tr>
<td>QA_Min_Task_Duration</td>
<td>-53/+15</td>
<td>+38/-22</td>
<td>-29/+19</td>
</tr>
<tr>
<td>RW_Min_Task_Duration</td>
<td>-27/+13</td>
<td>0/0</td>
<td>-15/+9</td>
</tr>
<tr>
<td>Task_List</td>
<td>-4/+11</td>
<td>-52/+53</td>
<td>-5/+5</td>
</tr>
<tr>
<td>Internal Precedence Relationship</td>
<td>-65/+5</td>
<td>+37/-10</td>
<td>-26/+19</td>
</tr>
<tr>
<td>Basic_prob_flawed_Task</td>
<td>-78/+18</td>
<td>-117/+41</td>
<td>-37/+18</td>
</tr>
</tbody>
</table>

**Table 3-4: Single Phase Sensitivity Test Results Project Performance in Percent Change from Likely Scenario under Pessimistic and Optimistic Scenarios (partial)**
<table>
<thead>
<tr>
<th>Parameter Name</th>
<th>Project Cycle Time Performance Change (%)</th>
<th>Project Quality Performance Change (%)</th>
<th>Project Cost Performance Change (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Resources:</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>BW_Priority</td>
<td>0/0</td>
<td>0/0</td>
<td>0/0</td>
</tr>
<tr>
<td>QA_Priority</td>
<td>-7/-2</td>
<td>-4/+25</td>
<td>+5/-4</td>
</tr>
<tr>
<td>RW_Priority</td>
<td>0/+2</td>
<td>0/-7</td>
<td>0/+1</td>
</tr>
<tr>
<td>BW_Labor_Delay</td>
<td>-7/-4</td>
<td>+17/-11</td>
<td>-4/+10</td>
</tr>
<tr>
<td>QA_Labor_Delay</td>
<td>-7/+11</td>
<td>+2/+47</td>
<td>-21/-10</td>
</tr>
<tr>
<td>RW_Labor_Delay</td>
<td>0/0</td>
<td>0/0</td>
<td>0/0</td>
</tr>
<tr>
<td>Max_Headcount</td>
<td>+2/0</td>
<td>-1/0</td>
<td>+49/-31</td>
</tr>
<tr>
<td>Headcount_Adjustment_Time</td>
<td>0/0</td>
<td>+2/-2</td>
<td>-25/+12</td>
</tr>
<tr>
<td>Initial_Headcount</td>
<td>-35/+2</td>
<td>-7/-2</td>
<td>+30/-3</td>
</tr>
<tr>
<td>Exper_Assim_Time</td>
<td>0/0</td>
<td>0/0</td>
<td>-1/0</td>
</tr>
<tr>
<td><strong>Targets and Performance:</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Initial_Proj_Deadline</td>
<td>+7/-9</td>
<td>+7/-10</td>
<td>-6/+13</td>
</tr>
<tr>
<td>Project_Quality_Goal</td>
<td>0/0</td>
<td>0/0</td>
<td>0/0</td>
</tr>
<tr>
<td>Project Budget (X1000)</td>
<td>0/0</td>
<td>+2/0</td>
<td>+19/-10</td>
</tr>
<tr>
<td>Quality_Goal_Adjust_Time</td>
<td>0/0</td>
<td>0/0</td>
<td>0/0</td>
</tr>
<tr>
<td>Resistance_to_Sched_Slip</td>
<td>-4/-9</td>
<td>-2/-9</td>
<td>+3/+14</td>
</tr>
</tbody>
</table>

**Table 3-4: Single Phase Sensitivity Test Results Project Performance in Percent Change from Likely Scenario under Pessimistic and Optimistic Scenarios (continued)**

The sensitivity of the model behavior is the range of performance change (in percent of likely scenario performance). These results are shown in Table 3-5. For example, the 53% decrease in schedule performance and 15% increase in schedule performance for the previously described example produce a 68% total sensitivity of the model's schedule performance to the Quality Assurance Minimum Task Duration parameter. This value is shown in the left data column and second row of Table 3-5.
<table>
<thead>
<tr>
<th>Parameter Name</th>
<th>Project Cycle Time Performance Range (%)</th>
<th>Project Quality Performance Range (%)</th>
<th>Project Cost Performance Range (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Process Structure and Scope:</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>BW_Min_Task_duration</td>
<td>22</td>
<td>10</td>
<td>32</td>
</tr>
<tr>
<td>QA_Min_Task_Duration</td>
<td>68</td>
<td>50</td>
<td>48</td>
</tr>
<tr>
<td>RW_Min_Task_Duration</td>
<td>40</td>
<td>0</td>
<td>24</td>
</tr>
<tr>
<td>Task_List</td>
<td>15</td>
<td>105</td>
<td>10</td>
</tr>
<tr>
<td>Internal Precedence Relationship</td>
<td>70</td>
<td>47</td>
<td>45</td>
</tr>
<tr>
<td>Basic_prob_flawed_Task</td>
<td>96</td>
<td>158</td>
<td>55</td>
</tr>
<tr>
<td><strong>Resources:</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>BW_Priority</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>QA_Priority</td>
<td>5</td>
<td>29</td>
<td>9</td>
</tr>
<tr>
<td>RW_Priority</td>
<td>2</td>
<td>7</td>
<td>1</td>
</tr>
<tr>
<td>BW_Labor_Delay</td>
<td>3</td>
<td>28</td>
<td>14</td>
</tr>
<tr>
<td>QA_Labor_Delay</td>
<td>18</td>
<td>49</td>
<td>31</td>
</tr>
<tr>
<td>RW_Labor_Delay</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Max_Headcount</td>
<td>2</td>
<td>1</td>
<td>80</td>
</tr>
<tr>
<td>Headcount_Adjustment_Time</td>
<td>0</td>
<td>4</td>
<td>37</td>
</tr>
<tr>
<td>Initial_Headcount</td>
<td>37</td>
<td>5</td>
<td>33</td>
</tr>
<tr>
<td>Exper_Assim_Time</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td><strong>Targets and Performance:</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Initial_Proj_Deadline</td>
<td>16</td>
<td>17</td>
<td>19</td>
</tr>
<tr>
<td>Project_Quality_Goal</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Project Budget (X1000)</td>
<td>0</td>
<td>2</td>
<td>29</td>
</tr>
<tr>
<td>Quality_Goal_Adjust_Time</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Resistance_to_Sched_Slip</td>
<td>13</td>
<td>7</td>
<td>17</td>
</tr>
</tbody>
</table>

Table 3-5: Single Phase Sensitivity Test Results
Total Sensitivity to Parameter Change

The parameters to which the model behavior is most sensitive depends on the performance measure used. Based on the range of performance changes in Table 3-5, the three performance measures are most sensitive to the following ten parameters:

- **Cycle Time**: Basic probability that a Task is Flawed (96%), Internal Precedence Relationship (70%), Quality Assurance Minimum Task Duration (68%), Rework Minimum Task Duration (38%)

- **Quality**: Basic probability that a Task is Flawed (158%), Task List (105%), Quality Assurance Minimum Task Duration (50%), Quality Assurance Labor Delay (49%), Internal Precedence Relationship (47%)
• **Cost:** Maximum Headcount (80%), Basic probability that a Task is Flawed (55%), Quality Assurance Minimum Task Duration (48%), Internal Precedence Relationship (45%), Headcount Adjustment Time (37%),

If the three measures of performance were valued equally (unlikely in practice) the descending order of the four parameters which the model is most sensitive to would be: Basic probability that a Task is Flawed, Quality Assurance Minimum Task Duration, Internal Precedence Relationship, and Maximum Headcount.

### 3.5.4 Multiple Phase Model Sensitivity

A three-phase model was built to test the sensitivity of the multiple phase model to parameter values. The phase network is shown in Figure 3-33. Six parameters were added to those used to test the sensitivity of the single phase model. Three of those parameters represent the coordination activity, which is not active in a single phase model. These coordination parameters were the Coordination Minimum Task Duration, Coordination Priority and the Coordination Labor Delay. The other three parameters are the three External Precedence Relationships between the three phases in the test model.

![Multiple Phase Model for Sensitivity Testing](image)

**Figure 3-33: Multiple Phase Model for Sensitivity Testing**
The sensitivity tests described in the previous section for a single phase test model were repeated for the multiple phase test model using the parameter values shown in Table 3-6. The External Precedence Relationships are shown and described with the description of the Development Tasks sector.

<table>
<thead>
<tr>
<th>Parameter Name</th>
<th>Pessimistic Scenario Value</th>
<th>Likely Scenario Value</th>
<th>Optimistic Scenario Value</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Process Structure and Scope:</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>BW_Min_Task_duration</td>
<td>4</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>QA_Min_Task_Duration</td>
<td>3</td>
<td>1</td>
<td>0.5</td>
</tr>
<tr>
<td>RW_Min_Task_Duration</td>
<td>2</td>
<td>1</td>
<td>0.5</td>
</tr>
<tr>
<td>Coord_Min_Task_duration</td>
<td>3</td>
<td>1</td>
<td>0.5</td>
</tr>
<tr>
<td>Task_List</td>
<td>1500</td>
<td>1000</td>
<td>500</td>
</tr>
<tr>
<td>Internal Precedence Relationships</td>
<td>linear</td>
<td>hyperbolic</td>
<td>open</td>
</tr>
<tr>
<td>Ext. Precedence Relationship:1-2</td>
<td>sequential</td>
<td>&quot;S&quot;</td>
<td>parallel</td>
</tr>
<tr>
<td>Ext. Precedence Relationship:1-3</td>
<td>sequential</td>
<td>lockstep</td>
<td>parallel</td>
</tr>
<tr>
<td>Ext. Precedence Relationship:2-3</td>
<td>sequential</td>
<td>&quot;S&quot;</td>
<td>parallel</td>
</tr>
<tr>
<td>Basic_prob_flawed_Task</td>
<td>0.9</td>
<td>0.5</td>
<td>0.3</td>
</tr>
<tr>
<td><strong>Resources:</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>BW_Priority</td>
<td>6</td>
<td>3</td>
<td>1.5</td>
</tr>
<tr>
<td>QA_Priority</td>
<td>0.2</td>
<td>1</td>
<td>5</td>
</tr>
<tr>
<td>RW_Priority</td>
<td>0.2</td>
<td>1</td>
<td>5</td>
</tr>
<tr>
<td>Coord_Priority</td>
<td>0.2</td>
<td>1</td>
<td>5</td>
</tr>
<tr>
<td>BW_Labor_Delay</td>
<td>6</td>
<td>2</td>
<td>0.5</td>
</tr>
<tr>
<td>QA_Labor_Delay</td>
<td>16</td>
<td>8</td>
<td>2</td>
</tr>
<tr>
<td>RW_Labor_Delay</td>
<td>4</td>
<td>1</td>
<td>0.25</td>
</tr>
<tr>
<td>Coord_Labor_Delay</td>
<td>8</td>
<td>4</td>
<td>2</td>
</tr>
<tr>
<td>Max_Headcount</td>
<td>0.5</td>
<td>1</td>
<td>5</td>
</tr>
<tr>
<td>Headcount_Adjustment_Time</td>
<td>16</td>
<td>8</td>
<td>4</td>
</tr>
<tr>
<td>Initial_Headcount</td>
<td>0.1</td>
<td>0.5</td>
<td>1.0</td>
</tr>
<tr>
<td>Exper_Assim_Time</td>
<td>0.1</td>
<td>0.5</td>
<td>1.0</td>
</tr>
<tr>
<td><strong>Targets and Performance:</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Initial_Proj_Deadline</td>
<td>75</td>
<td>125</td>
<td>250</td>
</tr>
<tr>
<td>Project_Quality_Goal</td>
<td>1.0</td>
<td>0.9</td>
<td>0.7</td>
</tr>
<tr>
<td>Project Budget (x 1000)</td>
<td>375</td>
<td>500</td>
<td>1000</td>
</tr>
<tr>
<td>Quality_Goal_Adjust_Time</td>
<td>48</td>
<td>12</td>
<td>4</td>
</tr>
<tr>
<td>Resistance_to_Sched_Slip</td>
<td>0.5</td>
<td>2</td>
<td>5</td>
</tr>
</tbody>
</table>

Table 3-6: Multiple Phase Sensitivity Test Parameter Values

Three measures of project performance were used in the multiple phase sensitivity tests: total project cycle time, flawed tasks released from the three phases at the completion of
the project, and total project cost. The multiple phase sensitivity tests produced the results shown in Tables 3-7, 3-8, and 3-9.

<table>
<thead>
<tr>
<th>Parameter Name</th>
<th>Project Cycle Time Performance (weeks)</th>
<th>Project Quality Performance (Defects Released)</th>
<th>Project Cost Performance ($ x 1000)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Likely Scenario Performance</td>
<td>140</td>
<td>237</td>
<td>714</td>
</tr>
<tr>
<td>Process Structure and Scope:</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>BW_Min_Task_duration</td>
<td>154/126</td>
<td>237/239</td>
<td>770/641</td>
</tr>
<tr>
<td>QA_Min_Task_Duration</td>
<td>211/125</td>
<td>142/292</td>
<td>882/670</td>
</tr>
<tr>
<td>RW_Min_Task_Duration</td>
<td>183/121</td>
<td>225/169</td>
<td>821/660</td>
</tr>
<tr>
<td>Coord_Min_Task_duration</td>
<td>110/168</td>
<td>306/298</td>
<td>591/811</td>
</tr>
<tr>
<td>Task_List</td>
<td>141/142</td>
<td>286/168</td>
<td>730/702</td>
</tr>
<tr>
<td>Internal Precedence Relationships</td>
<td>177/128</td>
<td>159/289</td>
<td>851/658</td>
</tr>
<tr>
<td>Ext. Precedence Relationship:1-3</td>
<td>305/131</td>
<td>449/237</td>
<td>894/706</td>
</tr>
<tr>
<td>Ext. Precedence Relationship:2-3</td>
<td>320/138</td>
<td>333/202</td>
<td>585/703</td>
</tr>
<tr>
<td>Basic_prob_flawed_Task</td>
<td>351/120</td>
<td>434/194</td>
<td>1,011/644</td>
</tr>
<tr>
<td>Resources:</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>BW_Priority</td>
<td>149/138</td>
<td>253/182</td>
<td>741/704</td>
</tr>
<tr>
<td>QA_Priority</td>
<td>142/153</td>
<td>164/284</td>
<td>730/746</td>
</tr>
<tr>
<td>RW_Priority</td>
<td>140/136</td>
<td>237/197</td>
<td>714/709</td>
</tr>
<tr>
<td>Coord_Priority</td>
<td>140/140</td>
<td>238/239</td>
<td>715/716</td>
</tr>
<tr>
<td>BW_Labor_Delay</td>
<td>138/138</td>
<td>303/232</td>
<td>616/714</td>
</tr>
<tr>
<td>QA_Labor_Delay</td>
<td>216/79</td>
<td>301/132</td>
<td>985/407</td>
</tr>
<tr>
<td>RW_Labor_Delay</td>
<td>141/140</td>
<td>185/242</td>
<td>729/716</td>
</tr>
<tr>
<td>Coord_Labor_Delay</td>
<td>163/140</td>
<td>237/237</td>
<td>714/714</td>
</tr>
<tr>
<td>Max_Headcount</td>
<td>131/144</td>
<td>122/551</td>
<td>401/1,167</td>
</tr>
<tr>
<td>Headcount_Adjustment_Time</td>
<td>139/138</td>
<td>234/221</td>
<td>761/675</td>
</tr>
<tr>
<td>Initial_Headcount;</td>
<td>141/139</td>
<td>287/229</td>
<td>680/724</td>
</tr>
<tr>
<td>Exper_Assim_Time</td>
<td>142/139</td>
<td>243/234</td>
<td>725/709</td>
</tr>
<tr>
<td>Targets and Performance:</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Initial_Proj_Deadline</td>
<td>141/142</td>
<td>228/280</td>
<td>725/705</td>
</tr>
<tr>
<td>Project_Quality_Goal</td>
<td>140/140</td>
<td>235/237</td>
<td>715/714</td>
</tr>
<tr>
<td>Project Budget (X1000)</td>
<td>143/137</td>
<td>273/179</td>
<td>678/740</td>
</tr>
<tr>
<td>Quality_Goal_Adjust_Time</td>
<td>140/140</td>
<td>237/237</td>
<td>714/714</td>
</tr>
<tr>
<td>Resistance_to_Sched_Slip</td>
<td>141/139</td>
<td>201/311</td>
<td>699/688</td>
</tr>
</tbody>
</table>

Table 3-7: Multiple Phase Sensitivity Test Results

Project Performance under Pessimistic and Optimistic Scenarios
<table>
<thead>
<tr>
<th>Parameter Name</th>
<th>Project Cycle Time Performance Change (%)</th>
<th>Project Quality Performance Change (%)</th>
<th>Project Cost Performance Change (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Process Structure and Scope:</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>BW_Min_Task_duration</td>
<td>-10/+10</td>
<td>0/-1</td>
<td>-8/+10</td>
</tr>
<tr>
<td>QA_Min_Task_Duration</td>
<td>-51/+11</td>
<td>+40/-23</td>
<td>-25/+6</td>
</tr>
<tr>
<td>RW_Min_Task_Duration</td>
<td>-31/+13</td>
<td>+5/+28</td>
<td>-15/+7</td>
</tr>
<tr>
<td>Coord_Min_Task_duration</td>
<td>+21/-20</td>
<td>-29/-26</td>
<td>+17/-13</td>
</tr>
<tr>
<td>Task_List</td>
<td>-1/-1</td>
<td>-21/+29</td>
<td>-2/+2</td>
</tr>
<tr>
<td>Internal Precedence Relationships</td>
<td>-26/+8</td>
<td>+33/-22</td>
<td>-19/+8</td>
</tr>
<tr>
<td>Ext. Precedence Relationship:1-2</td>
<td>-73/+1</td>
<td>+25/+8</td>
<td>0/+3</td>
</tr>
<tr>
<td>Ext. Precedence Relationship:1-3</td>
<td>-118/+6</td>
<td>-89/0</td>
<td>-25/+1</td>
</tr>
<tr>
<td>Ext. Precedence Relationship:2-3</td>
<td>-128/+1</td>
<td>-40/+15</td>
<td>+18/+1</td>
</tr>
<tr>
<td>Basic_prob_flawed_Task</td>
<td>-151/+14</td>
<td>-83/+18</td>
<td>-41/+10</td>
</tr>
<tr>
<td><strong>Resources:</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>BW_Priority</td>
<td>-6/+1</td>
<td>-7/+23</td>
<td>-4/+1</td>
</tr>
<tr>
<td>QA_Priority</td>
<td>-1/-9</td>
<td>+31/-20</td>
<td>-2/-4</td>
</tr>
<tr>
<td>RW_Priority</td>
<td>0/+3</td>
<td>0/+17</td>
<td>0/+1</td>
</tr>
<tr>
<td>Coord_Priority</td>
<td>0/0</td>
<td>0/-1</td>
<td>0/0</td>
</tr>
<tr>
<td>BW_Labor_Delay</td>
<td>1/1</td>
<td>-28/+2</td>
<td>+14/0</td>
</tr>
<tr>
<td>QA_Labor_Delay</td>
<td>-54/+43</td>
<td>-27/+44</td>
<td>-38/+43</td>
</tr>
<tr>
<td>RW_Labor_Delay</td>
<td>-1/0</td>
<td>+22/-2</td>
<td>-2/0</td>
</tr>
<tr>
<td>Coord_Labor_Delay</td>
<td>-16/0</td>
<td>0/0</td>
<td>0/0</td>
</tr>
<tr>
<td>Max_Headcount</td>
<td>+6/-3</td>
<td>+48/-132</td>
<td>+44/-63</td>
</tr>
<tr>
<td>Headcount_Adjustment_Time</td>
<td>+1/+1</td>
<td>+1/+7</td>
<td>-6/+5</td>
</tr>
<tr>
<td>Initial_Headcount</td>
<td>-1/+1</td>
<td>-21/+3</td>
<td>+5/-1</td>
</tr>
<tr>
<td>Exper_Assim_Time</td>
<td>-1/+1</td>
<td>-2/+1</td>
<td>-1/+1</td>
</tr>
</tbody>
</table>

**Targets and Performance:**
- Initial_Proj_Deadline: -1/-1, +4/-18, -1/+1
- Project_Quality_Goal: 0/0, +1/0, 0/0
- Project_Budget (X1000): -2/+2, -15/+24, +5/+4
- Quality_Goal_Adjust_Time: 0/0, 0/0, 0/0
- Resistance_to_Sched_Slip: -1/+1, +15/-31, +2/+4

Table 3-8: Multiple Phase Sensitivity Test Results
Project Performance in Percent Change from Likely Scenario under Pessimistic and Optimistic Scenarios
<table>
<thead>
<tr>
<th>Parameter Name</th>
<th>Project Cycle Time Performance Range (%)</th>
<th>Project Quality Performance Range (%)</th>
<th>Project Cost Performance Range (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Process Structure and Scope:</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>BW_Min_Task_duration</td>
<td>20</td>
<td>1</td>
<td>18</td>
</tr>
<tr>
<td>QA_Min_Task_Duration</td>
<td>62</td>
<td>63</td>
<td>31</td>
</tr>
<tr>
<td>RW_Min_Task_Duration</td>
<td>44</td>
<td>23</td>
<td>22</td>
</tr>
<tr>
<td>Coord_Min_Task_duration</td>
<td>41</td>
<td>3</td>
<td>30</td>
</tr>
<tr>
<td>Task_List</td>
<td>0</td>
<td>50</td>
<td>4</td>
</tr>
<tr>
<td>Internal Precedence Relationships</td>
<td>34</td>
<td>55</td>
<td>27</td>
</tr>
<tr>
<td>Ext. Precedence Relationship:1-2</td>
<td>74</td>
<td>17</td>
<td>3</td>
</tr>
<tr>
<td>Ext. Precedence Relationship:1-3</td>
<td>124</td>
<td>89</td>
<td>26</td>
</tr>
<tr>
<td>Ext. Precedence Relationship:2-3</td>
<td>129</td>
<td>55</td>
<td>17</td>
</tr>
<tr>
<td>Basic_prob_flawed_Task</td>
<td>165</td>
<td>101</td>
<td>51</td>
</tr>
<tr>
<td>Resources:</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>BW_Priority</td>
<td>7</td>
<td>30</td>
<td>5</td>
</tr>
<tr>
<td>QA_Priority</td>
<td>8</td>
<td>51</td>
<td>2</td>
</tr>
<tr>
<td>RW_Priority</td>
<td>3</td>
<td>17</td>
<td>1</td>
</tr>
<tr>
<td>Coord_Priority</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>BW_Labor_Delay</td>
<td>0</td>
<td>30</td>
<td>14</td>
</tr>
<tr>
<td>QA_Labor_Delay</td>
<td>97</td>
<td>71</td>
<td>81</td>
</tr>
<tr>
<td>RW_Labor_Delay</td>
<td>1</td>
<td>24</td>
<td>2</td>
</tr>
<tr>
<td>Coord_Labor_Delay</td>
<td>16</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Max_Headcount</td>
<td>9</td>
<td>180</td>
<td>107</td>
</tr>
<tr>
<td>Headcount_Adjustment_Time</td>
<td>0</td>
<td>6</td>
<td>11</td>
</tr>
<tr>
<td>Initial_Headcount</td>
<td>2</td>
<td>24</td>
<td>6</td>
</tr>
<tr>
<td>Exper_Assim_Time</td>
<td>2</td>
<td>3</td>
<td>2</td>
</tr>
</tbody>
</table>

Table 3-9: Multiple Phase Sensitivity Test Results
Total Sensitivity to Parameter Change

As in the single-phase sensitivity tests, the multiple phase model is most sensitive to different parameters depending on the performance measure used. Based on the range of performance changes in Table 3-9, the three performance measures are most sensitive to the following six parameters:
• **Project Cycle Time**: Basic probability that a Task is flawed (165%), External Precedence Relationship: phase 2 to phase 3 (129%), External Precedence Relationship: phase 1 to phase 3 (124%), Quality Assurance Labor Delay (97%), and Quality Assurance Minimum Task Duration (62%).

• **Project Quality**: Maximum Headcount (180%), Basic probability that a Task is flawed (101%), External Precedence Relationship: phase 1 to phase 3 (89%), Quality Assurance Labor Delay (71%), and Quality Assurance Minimum Task Duration (63%).

• **Project Cost**: Maximum Headcount (107%), Quality Assurance Labor Delay (81%), Basic probability that a Task is flawed (51%), and Quality Assurance Minimum Task Duration (31%).

If the three measures of performance were valued equally (unlikely in practice), the descending order of the five parameters which the model is most sensitive to would be: Basic probability that a Task is Flawed, Maximum Headcount, Quality Assurance Labor Delay, two External Precedence Relationships, and Quality Assurance Minimum Task Duration.

### 3.5.5 Model Sensitivity Test Summary

Sensitivity tests on a single phase test model and a multiple phase model identified parameters which have relatively large influence on model behavior. Model behavior was measured in the three primary dimensions of project performance: cycle time, quality, and cost. All three performance measures are sensitive to the basic rate of error generation, with quality performance being more sensitive than cycle time and cost. This parameter can be seen as a measure of task difficulty or newness to the developers. The high leverage which this parameter has on performance helps explain the challenges of developing new and more complex products.

The parameters to which the model's behavior is most sensitive is relatively consistent between the single phase and multiple phase tests. Approximately half of the high leverage parameters for both the single and multiple phase test models describe the development process. This supports the need for process structure components in dynamic models of projects. All but one of the six performance measures were sensitive to the precedence relationships which describe the levels of process concurrence. This
indicates that designing and managing concurrence in development may be a high leverage point for improved performance. Inter-phase relationships overshadowed intra-phase relationships in the multiple phase test. This may indicate that macro-process design and improvement is a higher leverage point than micro-process design and improvement. A second process descriptor, the Quality Assurance Task Duration also influenced five of the six performance measures. This combines with the important role of the Basic probability of a Flawed Task parameter to suggest that quality improvement efforts can be effective at improving development project performance.

In contrast to the process description parameters, none of the high leverage parameters are targets. This implies that setting aggressive project goals is not as effective at improving performance as addressing some other parameters.

Differences in the influence of high leverage parameters on performance measures appear as would be expected. For example Maximum Headcount influenced cost performance more than quality or cycle time. Available work constraints influenced cycle time more often than quality and cost. Quality assurance and rework parameters influenced the quality measure most.

The sensitivity tests identify parameters which deserve particular attention in the estimation of parameter values because of their impacts on simulation results. The results also indicate potentially powerful areas for system design and improvement.

3.6 Model Description Summary

The Product Development Project Model simulates from one to five development phases within a single project. A project phase network links individual development phases through available work constraints and error flows. Targets and performance are measured in three dimensions: cycle time, quality and cost. They are modeled at the project level and also link individual phases. Each development phase models process structure, scope, resources, targets, and performance. The process structure and scope
subsystems model development task flows as well as internal, inherited and released errors. Resources subsystem sectors model the quantity, effectiveness and allocation of labor among four development activities. Performance relative to cycle time, quality, and cost targets impact labor allocation, workweek length, and headcount.

Sensitivity tests indicate that model behavior for the three performance measures for single and multiple phase test models is most sensitive to development process parameters, especially task difficulty, precedence relationships and quality assurance parameters. These parameters will be addressed in more detail in Chapter 5: The Python Development Project.
Chapter 4

A Signal Processing Model of a Product Development Process Phase

4.1 Introduction

Multiple perspectives can facilitate understanding a model, its limitations and uses, and the system it describes. In this chapter I describe my application of an alternative modeling approach to a portion of the Product Development Project Model. This work extends the investigation of model behavior and illustrates a different and valuable means of understanding system behavior. This improved understanding can aid in the design of rigorous policies, practices, and systems.

I built a signal processing model of some of the most important parts of the Development Tasks sector (Figure 3-8) for a single phase. The system was analyzed to determine the conditions required for stable system behavior and system-induced oscillation. The model was used to generate simulations across a range of parameter values to improve understanding of the impacts of individual parameters on system behavior. Finally, the
limitations, use, and potential applications of the signal processing modeling approach is discussed.

4.2 System Descriptions

The system dynamics and signal processing representations of the system modeled in this chapter are shown in Figures 4-1 and 4-2.

![Figure 4-1: A System Dynamics Representation of a Product Development Process Phase Model](image_url)

- Known Rework
- Inspection Failure
- INSPECTION DELAY
- Completed not Inspected
- Tasks Released
- Basework
- Rework Tasks
- Release Tasks

IFF: Inspection Failure Fraction
Figure 4-2: A Signal Processing Representation of a Product Development Process Phase Model

These two system representations are mathematically equivalent. The following differential equations describe the system:

**The State Variables:**

\[
CT_i = CT_{i0} + \int_0^T (BW_i + RW_i - IF_i - RT_i) \, dt
\]

where
- \(i\) = development phase
- \(T\) = time, weeks
- \(CT\) = Completed but not Checked, tasks
- \(BW\) = Basework rate, tasks per week
- \(RW\) = Rework rate, tasks per week
- \(IF\) = Rework due to Inspection Failures, tasks per week
- \(RT\) = Release Tasks rate, tasks per week
\[ KRW_i = KRW_{i0} + \int_0^T (IF_i - RW_i) \, dt \]

where

- \( KRW \) = Know Rework, tasks
- \( IF \) = Rework due to Inspection Failures, tasks per week
- \( RW \) = Rework rate, tasks per week

\[ TR_i = TR_{i0} + \int_0^T RT_i \, dt \]

where

- \( TR \) = Tasks Released, tasks
- \( RT \) = Release Tasks rate, tasks per week

**The Transfer Equations:**

\[ RW_i = \frac{KRW}{RWDur} \]

where

- \( RW \) = Rework rate, tasks per week
- \( KRW \) = Know Rework, tasks
- \( RWDur \) = Minimum Rework Task Duration, weeks

\[ IF_i = IFF \times \left( \frac{CT}{QADur} \right) \]

where

- \( IF \) = Rework due to Inspection Failures, tasks per week
- \( IFF \) = Inspection Failure Fraction, dimensionless
- \( CT \) = Completed, not Checked stock, tasks
- \( QADur \) = Quality Assurance Minimum Duration, week
\[ RT_1 = (1 - IFF) \times (CT / QADur) \]

where
- \( RT \) = Release Tasks rate, tasks per week
- \( CT \) = Completed, not Checked stock, tasks
- \( QADur \) = Quality Assurance Minimum Duration, week
- \( IFF \) = Inspection Failure Fraction, dimensionless

The basis for these equations is described in chapter 3. The equations describing the signal processing model (Figure 4-2) can be rearranged into a system description in the standard matrix form:

\[
\frac{dx}{dt} = A \times x + B \times f
\]

\[ y = C \times x + D \times f \]

where
- \( x \) - a vector of state variables
- \( \frac{dx}{dt} \) - the rate of change of the state variables over time
- \( y \) - output signal, taken to be the Tasks Released state variable
- \( f \) - input (forcing) signal, taken to be the Basework rate
- \( A \) - system matrix
- \( B, C, D \) - matrices describing the relationships among the input signal and rate of change function or output signal

For the model shown the matrices are:

\[
A = \begin{bmatrix}
-1/T1 & 1/T2 & 0 \\
IFF/T1 & -1/T2 & 0 \\
(1-IFF)/T1 & 0 & 0
\end{bmatrix}
\]

\[
B = \begin{bmatrix}
BW & 0 & 0
\end{bmatrix}
\]

\[
C = \begin{bmatrix}
0 & 0 & 1
\end{bmatrix}
\]

\[
D = \begin{bmatrix}
0 & 0 & 0
\end{bmatrix}
\]
The preceding equations were programmed into a signal processing simulation package (Burras et al., 1994) with forcing functions (input signals) and system responses (outputs) for analysis and simulation. The code is shown in Appendix 4.1

4.3 System Behavior

4.3.1 Introduction

Knowing the parameter values which generate stability of system behavior is important for understanding a system. A large body of systems analysis methodology exists for studying systems described in the signal processing form above (e.g. see Dorf and Bishop, 1995; Ogata, 1989; Oppenheim and Schafer, 1989). Many methods of investigating behavioral stability are available from this domain. A fundamental approach is used for the illustrative purposes of this chapter. The conditions for system stability were investigated using eigenvalues of the system matrix. This approach has previously been successfully applied to a system dynamics model of an economy (Forrester, 1982). The single phase process system has three eigenvalues:

\[
0 \text{ (input determines behavior)}
\]

\[
\left( \frac{1}{2} \right) \left( -T_2 - T_1 + \sqrt{T_2^2 - 2 \cdot T_1 \cdot T_2 + T_1^2 + 4 \cdot T_1 \cdot T_2 \cdot IF \right. \over T_1 \cdot T_2}
\]

\[
\left( \frac{1}{2} \right) \left( -T_2 - T_1 - \sqrt{T_2^2 - 2 \cdot T_1 \cdot T_2 + T_1^2 + 4 \cdot T_1 \cdot T_2 \cdot IF \right. \over T_1 \cdot T_2}
\]

4.3.2 Conditions for Stability

System eigenvalues define a system's poles in the frequency domain. Those poles can be used to describe system behavior by plotting them in a root locus plot in the S-plane. The
S-plane is a Cartesian coordinate system with real components plotted on along the abscissa and imaginary components plotted along the ordinate. Positive real system pole components correspond to unstable system behavior. Negative real pole components correspond to stable system behavior. A more detailed discussion of the use of the frequency domain is available in Karu (1995) or Oppenheim and Schafer (1989).

The two non-zero system poles can be simplified for analysis into the sum of or difference between two expressions, referred to here as Expr1 and Expr2:

\[
\frac{-T2-T1}{2*T1*T2} + \sqrt{\frac{T2^2 - 2*T1*T2 + T1^2 + 4*T1*T2*IFF}{2*T1*T2}}
\]

Expr1    Expr2

The conditions for potential system instability can be shown by plotting Expr1 and Expr2 on the S-plane (Figure 4-3).

![Figure 4-3: The System Poles Plotted in the S-Plane](image-url)
From Figure 4-3 the two potential causes of system unstable behavior are:

- $\text{Expr1} > 0$
- $\text{Expr2} > (-1) \times \text{Expr1}$

The first potential condition for instability reduces to

$$\text{Expr1} = \frac{-T2-T1}{2 \times T1 \times T2} = \frac{-1}{2} \times \frac{T2+T1}{T1 \times T2} > 0 \text{ (unstable system)}$$

or

$$\frac{T2+T1}{T1 \times T2} < 0$$

By inspection the preceding inequality can only occur when $T1 < 0$ or $T2 < 0$.

A central part of the traditional signal processing approach to system evaluation is the consideration of system behavior under all parameter values regardless of the impossibility or reasonableness of the values and the subsequent limiting of the model or parameter values based on the results of the evaluation. This evaluate-all-then-limit approach distinguishes the signal processing approach from the system dynamics approach which limits parameter values to possible and reasonable ranges and then evaluates system behavior within those limits.

An understanding of the meaning and significance of parameter values which can cause instability will allow reasonable restrictions on parameter ranges. The first conditions to be considered are the conditions just established for system in stability, when $T1 < 0$ or $T2 < 0$. $T1$ and $T2$ represent smoothing and delays of inspection and rework activities, respectively. Values of $T1 < 0$ or $T2 < 0$ would represent infeasible system conditions and cause impossible movement of development tasks such as reversing the inspection process, causing Known Rework to become "un-inspected" and return to the Completed but not Inspected stock. The results of such conditions can be shown with a simulation of
the system response to a pulse loading at time $t = 5$ and $T_1 = -10$, as shown in Figure 4-4. A single phase project system producing a negative number of Tasks Released is clearly infeasible.

![Figure 4-4: System Response (Tasks Released) to Pulse Signal with $T_1 = -10$](image)

Similar reasoning can be applied to the conditions in which $T_1 = 0$ or $T_2 = 0$. The condition $T_1 = 0$ implies that inspection begins instantaneously upon completion of Basework or Rework and takes no time to perform. This is not feasible in real systems. Likewise when $T_2 = 0$ Rework begins instantaneously upon identification of errors and takes no time to perform. This is also not feasible in real systems.

Based on the preceding the following restrictions and conclusions can be drawn concerning parameter values for $T_1$ and $T_2$ based on the first condition for stability:

- For a stable system behavior restrict Parameter Values for $T_1$ and $T_2$ to $T_1 > 0$ and $T_2 > 0$.
- No feasible system descriptions are eliminated by these restrictions.
- For $T_1 > 0$ and $T_2 > 0$ the system never meets the first condition for instability.
From Figure 4-3 the second potential cause of system instability is:

\[ \text{Expr2} > (-1) \times \text{Expr1} \]

This condition can be reduced to

\[ \frac{\sqrt{T2^2 - 2 \times T1 \times T2 + T1^2 + 4 \times T1 \times T2 \times \text{IFF}}}{2 \times T1 \times T2} > (-1) \times \frac{1}{2} \times \frac{T2 + T1}{T1 \times T2} \]

or

\[ \frac{\sqrt{T2^2 - 2 \times T1 \times T2 + T1^2 + 4 \times T1 \times T2 \times \text{IFF}}}{T2 + T1} > 1 \]

This condition uses the Inspection Failure Fraction (IFF). Three possible ranges of values are possible for IFF:

- \( \text{IFF} < 0 \)
- \( 0 \leq \text{IFF} \leq 1 \)
- \( \text{IFF} > 1 \)

An evaluation of the meaning and significance of IFF at the first and third ranges simplify the analysis. By definition IFF is a fractional amount of a non-negative quantity (tasks inspected) and remains within the range 0 - 1. Therefore IFF < 0 or IFF > 1 is meaningless and will not occur in actual systems.

Therefore the following restrictions and conclusions can be drawn concerning the values of IFF:

- Restrict Parameter Values for IFF to 0 \( \leq \) IFF \( \leq \) 0
- No feasible system descriptions are eliminated by these restrictions.
The second condition for instability can be evaluated using the prescribed range for IFF.

When IFF = 0 then system is unstable if \( \frac{T2 - T1}{T2 + T1} > 1 \)

By inspection this inequality can never be true within the prescribed ranges for T1 and T2 (T1 > 0 and T2 > 0).

When IFF = 1 then \( \frac{T2 + T1}{T2 + T1} = 1 > 1 \)

This inequality can never be true.

In the range 0 < IFF < 1 the left hand side of the inequality increases from

\[ \frac{T2 - T1}{T2 + T1} \text{ to } \frac{T2 + T1}{T2 + T1} = 1. \]

Therefore this range of values for IFF will never generate a true inequality when T1 > 0 and T2 > 0.

Based on the preceding the following conclusions can be drawn concerning the second condition for system instability.

- Restrict Parameter Values for IFF to 0 <= IFF <= 1.
- No feasible system descriptions are eliminated by this restriction.
- For 0 <= IFF <= 0 the system never meets the second condition for instability.

By combining the conclusions concerning the conditions for system instability the following conclusions can be drawn:

- The System is stable within parameter ranges T1 > 0, T2 > 0, 0 <= IFF <= 1.
- No feasible system descriptions are excluded by these parameter restrictions.
4.3.3 System Changes Which Impact Stability

The analysis of system stability prompts the question "What system changes could cause instability?" Two possibilities are suggested.

• **Increase system gain with IFF > 1:** If the assumption of tasks as atomic pieces of work and therefore completely correct or completely flawed is relaxed then more than one error could be generated in each task. This could be represented by IFF > 1. This could generate more rework than tasks, potentially increasing total work infinitely.

• **Add exogenous work:** If errors generated rework from beyond the project scope additional work could enter the system directly into the Known Rework stock and increase the work infinitely.

• **Describe the system with pure delays:** If the first order exponential delays currently used to describe the demand-based flows are replaced with infinite order delays which shift conditions without changing their values the system will behave significantly differently. Although this change is not sufficient alone to cause instability, it may contribute to instability when combined with other changes.

Identifying system changes which could cause important behavioral changes can expand the understanding of the model by investigating the impacts of stretching the model to describe new or different systems and conditions. Considering how the model behaves when stretched beyond its original envelope of assumptions and parameter values improves the understanding of the model within its original envelope.

4.3.4 Conditions for System Induced Oscillation

A second system behavior of possible interest is oscillatory behavior generated by the system. A system which induces oscillatory behavior is described in the frequency domain on the S-plane (Figure 4-3) by system poles plotted off the real component axis. This condition requires the eigenvalues to have imaginary components. This is true when:

\[ T2^2 - (2*T1*T2) + T1^2 + (4*T1*T2*IFF) < 0 \]
By inspection the minimum value of the left side of the inequality occurs when $\text{IFF} = 0$. Under this condition the inequality reduces to:

$$(T2 - T1)^2 < 0$$

This inequality can never be true since $T2$ and $T1$ are real numbers. Because the inequality can never be true the system eigenvalues cannot have imaginary components. Therefore:

- The system will not generate oscillation under feasible parameter values.

### 4.3.5 Typical System Behavior

Visualizing the system behavior within the prescribed parameter value ranges can increase understanding of system behavior. Two types of simulations will be used to illustrate system behavior:

- Simulation of the behavior of several system components over time under single specific parameter values. This helps improve understanding of the interaction of system components.
- Simulation of a single system component behavior over time across a range of parameter values. This helps improve understanding of the impacts of specific parameters on system behavior.

The use of the Release Tasks rate as a primary system output can facilitate improved understanding of system behavior. This is because as the rate of final completion of tasks the Release Tasks rate is directly analogous and comparable to the input signal, Basework, which is the initial completion of tasks.

#### 4.3.5.1 System Response to Impulse Signal

The impulse signal is a standard test of system behavior and produces the system's response to a single shock. Figures 4-5, 4-6, 4-7, and 4-8 show the behavior of several system components over time for an impulse input signal.
Input signal: pulse at time $t = 5$
$T1 = 5$
$T2 = 8$
$IFF = 25\%$

Figure 4-5: System Response (Tasks Released) to Impulse Signal

Figure 4-6: System Response (Known Rework) to Impulse Signal
Figure 4-7: System Response (Tasks Completed but not Checked) to Impulse Signal

Figure 4-8: System Response (Task Release Rate) to Impulse Signal
4.3.5.2 System Response to Block Signal

Another common test signal is the step signal, which changes instantaneously from one stable amplitude to another stable amplitude. However the pure step differs from all possible basework signals for a project model in at least one important way: their cumulative size. The step signal continues infinitely and therefore has an infinite cumulative size. But the initial completion of project tasks eventually returns to and remains zero. This limits the project scope to some real finite size.\(^4.1\) A similar but more realistic and useful signal for analyzing the behavior of a project model is a block signal. A block signal steps up from zero to a stable amplitude for a specific period of time and then steps down to zero again. The block signal is useful for testing and understanding a project model because it represents the limited scope of the project, whereas a pure step signal has no such limit. The block signal can represent an idealized project with a steady rate of initial task completion. The eventual return of the input signal (initial completions) to zero raises interesting and potentially important questions about projects:

- How does the system affect project performance after the signal has stopped?
- How long after the signal has stopped does it take for the project to end?

These and other questions can be investigated with a block test signal more easily than with a pure step signal. An additional tool for developing system understanding has been added to the investigation of system responses to a block signal: the visualization of changes in system responses as parameters change. Figure 4-9 illustrates this model investigation tool by showing the system response to a block signal over a range of values for the Inspection Time Constant, T1. The input signal and parameter values are:

Input signal: step from 0 to 10 at time t = 5 and step from 10 to 0 at time t = 45
T1 varies from 1 to 10
T2 = 8
IFF = 25%

\(^{4.1}\) The end to the basework signal and resulting existence of a limit on size is a fundamental distinction between projects, which end, and continuous operations, which are assumed to not end. In other words, a project must eventually come to an end or it is not a project.
Figure 4-9: System Response (Task Release Rate) to Block Signal
Impact of Inspection Time Constant

The increase in the time required for the Task Release Rate to reach steady conditions as the Inspection Time Constant increases is clear in figure 4-9. In contrast changes in the system response to the same signal as the Inspection Failure Fraction varies across its range from 0 to 1 is shown in Figure 4-10. The input signal and parameter values are:

- Input signal: step from 0 to 10 at time $t = 5$ and step from 10 to 0 at time $t = 45$
- $T1 = 5$
- $T2 = 8$
- IFF varies from 0 to 1
Figures 4-9 and 4-10 illustrate the different impacts of parameters on the system’s behavior and how the signal processing model facilitates understanding those impacts. This knowledge can be useful in the design of system changes which would alter parameter values and generate improved system responses.

4.4 Discussion of the Signal Processing Modeling Approach

4.4.1 Alternative System Descriptions and Analysis Tools

The illustrative examples herein and of applications of classical control theory to system dynamics (e.g. Kampmann, 1992, N. Forrester, 1982) demonstrate the potential benefits of applying a signal processing perspective to the Product Development Project Model. Those benefits include:

- Methods for the exhaustive exploration of system behavior under all parameter value ranges
- Analysis tools for the study of system behavior, particularly stability
- Tools for simulating and describing system behavior in very flexible ways which facilitate improved understanding of model behavior
4.4.2 Applicability of Linearity Assumptions

All modeling approaches are limited by their fundamental assumptions. One of the most important assumptions used in the signal processing approach is the use of only linear relationships. The domain of applicability of the approach is affected by this assumption.

Linear relationships appear reasonable and useful descriptions of several project decision rules. For example the linear equations describing the three demand-based flows of development tasks among the Completed but not Checked, Known Rework, and Tasks Released stocks appear reasonable. Linear descriptions may also be reasonable for some additional project process relationships. For example work availability constraints within a project phase may be linear in nature. An example is the first Internal Precedence Relationship described in Chapter 3 which describes the linear progression of installing the steel structure for a building.

However some important relationships in projects are nonlinear\textsuperscript{4.2}. For example the minimum of the work available and labor available limits as a basis for development activity rates is a fundamentally nonlinear relationship. Changes in the degree of concurrence of work within a phase as the phase progressed can cause nonlinear work-availability constraints. Inter-phase constraints may also be nonlinear. Other examples can be found in the relationships which describe the impacts of project conditions on humans such as the effect of schedule pressure on workweek. A graph of this relationship approaches asymptotic limits of the impacts and are nonlinear in their approaches. The nonlinearities in these relationships may be important determinants of system behavior. Describing some of these relationships with linear approximations may degrade the model's ability to describe the system's behavior (Forrester, 1992).

In conclusion the linear assumption used by the signal processing approach appears reasonable for many project relationships but limits the breadth of its applicability to

\textsuperscript{4.2} See Chapter 5 The Python Development Project for examples of nonlinear relationships.
modeling development projects. Careful consideration of each relationship between system components and the restrictions placed on parameter ranges allows the beneficial application of the signal processing approach to many portions of the Product Development Project Model.

4.4.3 Tractability of Analyzing Larger Systems

Another important issue is the tractability of analyzing larger systems with the signal processing approach. While systems significantly larger than the example given here can be analyzed with this approach the difficulty of analysis increases with system size and with the degree to which system components are linked. Closed form analysis becomes particularly challenging. This challenge leads to the use of two approaches: system decomposition and simulation. System decomposition seeks to model the system in separable linked pieces, each which can be more easily analyzed (Homer, 1983). The Product Development Project Model uses this approach by modeling a single project with a linked set of generic development phase structures. Hierarchical decomposition can extend this approach to analyzing models of larger systems. Simulation allows the investigation of models without the exhaustive analysis of each parameter (Clemson et al., 1995). These modeling approaches are useful for the analysis of models such as the complete Product Development Project Model.

4.5 Signal Processing Modeling Approach Summary

The signal processing approach was applied to a portion of the Product Development Project Model to investigate an alternative perspective to the traditional system dynamics modeling approach. Conditions for stable and oscillatory model behavior were studied using the system eigenvalues. The portion of the system studied was found to respond with stable non oscillatory behavior when restricted to parameter values which describe feasible project systems. System behavior was also investigated with simulations. These
Simulations illustrate system responses to an impulse signal and a block signal which can represent an idealized project. The use of visual displays of system responses as important parameter values vary was illustrated as one of the advantages of the signal processing approach. Issues raised by the application of the signal processing modeling approach were discussed, including the linear relationship assumption and the tractability of analyzing larger systems. The signal processing approach was found to provide several advantages and improve the analysis and understanding of the Product Development Project Model.
Chapter 5

The Python Development Project

5.1 Introduction

This chapter describes the calibration of the Product Development Project Model to a specific development project and its use to investigate a project management policy. The project will be referred to as the Python development project and International Chip Inc. (ICI) will refer to the organization responsible for the Python development project. The Python project developed a complex computer chip for a major player in the semiconductor industry. The project had several characteristics which made it useful for this research. Python was the first in a line of sequentially developed products. Therefore the project was relatively free of product family impacts (Wheelwright and Sasser, 1982). The Python chip was developed using product technology which ICI had used previously for several years. This eliminated technology research issues which would precede the development of the product itself (Iansiti, 1993a,b,c, 1992). The Python chip was also developed to be manufactured with mature production technology on facilities owned, controlled, and operated by ICI. This greatly minimized process development issues, keeping the Python project focused on the development of the product. Finally, the
Python development team was part of a single stable organization and almost completely immersed in a largely isolated development community with clearly established and documented development standards and strong cultural influences on development operations. This clarified and reinforced development project methodologies and decision-making routines. In summary, the Python project was consistent with the model boundary assumptions described in chapter 3 for the Product Development Project Model and provides a valuable case for the calibration and application of the model.

5.2 Data Collection

Data were collected concerning the computer chip development process, the ICI development process and organization, its development projects and the Python development team. The following methods and sources were used to collect data:

- **Internal corporate** publications provided data concerning the context of the product development process.
- **Interviews** were conducted with a cross functional team of product developers, managers of product development, and the majority of the Python development team\(^1\).
- **Workshops** were led with a new product introduction improvement team, a cross functional and cross product team formed to improve ICI’s product development process, the Python development team, and functionally-defined groups of developers.
- **Division records of aggregate product development process and performance data** were collected from the project control system.
- **Project specific products and records** were collected from individual developers.
- **Observations of development team operations** over 18 months improved my understanding of the practice of development at ICI and how it compares with the development plan.

\(^1\) The size of the Cobra development team is different depending on the source of data within ICI. Team size estimates range from approximately 18 to over three dozen. Thirty to thirty-five interviews were conducted.
These methods generated data at several levels of aggregation. At the development organization level data were collected concerning:

- The theoretical development process
- The development organization structure
- Aggregate project performance targets
- Aggregate project performance
- Product development improvement programs
- Product descriptions including complexity, markets, and functions

At the development team level data were collected concerning:

- Decision-making processes of managers and developers
- Deviations of development practice from the theoretical development process
- Resource descriptions including roles, responsibilities, availability, and interdependencies
- Process descriptions including constraints, relative difficulties of development activities and drivers of progress
- Development project objectives and priorities
- Development team organization, objectives, and processes

At the project level data were collected concerning:

- Development activity tasks, sizes, and products
- Movements of development tasks across time, organizational structures and process structure
- Resource loading
- Performance measures at the project and development phase levels
- Project development plans and targets
- Project objectives of developers
5.3 Computer Chip Development at ICI

5.3.1 A Generic Computer Chip Development Process

A typical computer chip development project passes through the activities below:

- Describe customer requirements
- Concept design
- Set metrics for product performance
- Design product subsystems
- Design product components
- Layout component locations for each chip layer
- Prepare Tape-out for mask making
- Fabricate masks for chip layers
- Fabricate test wafers
- Sort test wafers
- Assemble test chips into prototype products
- Test prototypes for functionality
- Test prototypes for manufacturability and release for full production
- Prepare customer support and launch information
- Launch product

The scopes, methods and durations for these activities vary widely depending on the product, development process and organization and development technology. For example the development of a simple memory chip may delete the design of subsystems completely. Development technologies also impact development. The layout design and drafting for chips was originally done manually. But the developers of complex chips now write computer code which facilitates the layout of thousands of components on many chip layers and the production of layout drawings. Typically several development
activities are aggregated for control purposes. For example ICI collects the development activities into seven development phases.

5.3.2 The Python Development Project Context

Product development at ICI is a focus of much attention. Successful development projects are essential to the organization's success. A detailed development plan is defined and described in several documents. Formal documents and standardized language are used to describe both the planned and actual development processes. This section will describe the formal development process and the important differences between theory and practice.

The product development process at ICI simultaneously addresses product development, business, and development project issues. Product development includes the description of product features, conceptual and detailed design, prototype fabrication and testing and design integration with manufacturing. Business issues include market studies and forecasting, profit and loss forecasts, financial metrics and product launch. Development project issues include project targets and metrics, resources, and management policies.

ICI is making large efforts to improve its product development and manufacturing performance. Many different and primarily independent improvement programs were simultaneously active at ICI at the time of the Python development project. These programs ranged in focus and direction from building and sharing a common vision for the world wide organization to increasing chip yield from wafers.

5.3.3 The Development Process

ICI uses a stage-gate development system (Cooper, R., 1994, 1990; O'Connor, 1994; Rosenthal, 1992) to describe their development process. Stage-gate systems apply process management to development by alternating development activities (stages) with
review and approval decisions (gates). Like many firms ICI has customized the stage-gate system for its own use. The system at ICI will be referred to as the ICI Development System or ICIDS. The stages are called phases. Phases typically have the name of a primary development activity. The ICIDS has eight phases:

- **Phase 1:** Discovery - Preliminarily describe the product, its place in the market, and the business opportunities and challenges

- **Phase 2:** Metrics - Validate Phase 1 with more detailed product description and business plan using business metrics. Set product performance metrics. Preliminarily plan the rest of the development project with cycle time and cost estimates.

- **Phase 3:** Product Definition - Fully describe the product technically including architecture, subsystems and interfaces. Refine business and development plans. Develop a launch plan.


- **Phase 5:** Fabrication and Assembly - Manufacture prototypes including test wafer fabrication, sorting and prototype assembly. Test prototypes for functionality defined in Phase 3: Product Definition specifications and alter specifications as required.

- **Phase 6:** Test, Characterization, and Pilot Production Test - Use prototypes to test product design for manufacturability and tune reliability and quality control of full production process.

- **Phase 7:** Launch - Prepare data for customer service. Offer product to customers and begin full production

- **Phase 8:** Process Improvement - Review product development project for continuous improvement.

ICI refers to the gate which follows each stage as a phase review. Although formal gate names differ from phase names the ICI organization typically refers to a phase review with the same number or name as the phase (e.g. "Phase 3 review"). The product of a phase review is the decision to take one of four actions on the development project:
• Kill the project
• Put the project on "hold", i.e. stop development work for an indefinite period of time
• Recycle the project to an earlier phase for additional work and review
• Approve the project to proceed to the next phase

Figure 5-1 shows ICI's stage-gate development process.
Although some of the phases in ICI's development process carry the names of development activities they are not the same as the model phases used in Chapter 3 to describe the Product Development Project Model. At ICI a phase is defined as all the development activities that occur between two phase review dates. For example Phase 4: Product Design is all the changes to the product description, marketing studies, business plan refinement, product design and prototype testing preparation which occurs after the Phase 2 review and before the Phase 3 review. In the Product Development Project Model a phase is all the work done by a functional development activity regardless of when during the project the work is performed (e.g. detailed design or prototype testing). Therefore model phases typically occur in several ICIDS phases. One challenge of this research was to distinguish between ICIDS phases and model phases. In this chapter "development phase" will refer to the ICIDS meaning (work between two dates) and "model phase" will refer to the Product Development Project Model meaning (a development activity).

A formal document describes the ICIDS and how it is intended to work. The document provides much detail about the activities and deliverables in each phase. But the uniqueness of product development project needs often require the developers to deviate from these detailed plans. Therefore the eight phases and phase reviews are the level of aggregation typically used by ICI to communicate about and manage development projects.

Product development as practiced at ICI follows the ICIDS plan conceptually but rarely if ever completely in the level of detail described in ICIDS documents. Some differences are inherent in the practice of development. For example, the process requires iteration even though the ICIDS is linear. In this way the ICIDS contrasts with some development plans which include inter-phase iteration (e.g. Peterson and Sutcliffe, 1992). The uniqueness of product development projects also requires the customization of the process plan on a project-by-project basis. The practice of phase reviews is an example of
the customizing which cause differences between the ICIDS as a plan and product development at ICI in practice. Some phase reviews are addressed much more formally than others. The Phase 1 and 2 reviews are typically informal meetings held by the members of the development team which have worked on the project to date (primarily marketing and product definition). In contrast the Phase 3 review (product definition) is considered one of the most important events in a development project at ICI. This is because passing the Phase 3 review initiates product design and a significant increase in the company's commitment of funds and labor to the project. Development projects are rarely killed at ICI, but Phase 3 reviews are postponed and projects are recycled at the Phase 3 review for additional work and re-review. The date of the Phase 3 review is also important in measuring the success of the project because one of the two primary cycle time metrics for the project starts on the date of the Phase 3 review. Phase 4 and 5 reviews are also informal, often being handled among the designers and testers without formal meetings. The signatures releasing the product to production defines the Phase 6 review. Product launches (phase 7) are often spread out over a period of time. Phase 8 and its review (Process Improvement) are rarely performed. In summary, ICI uses the ICIDS as a structure for its development process. Significant flexibility is allowed and used within that structure in the development of specific products.

5.3.4 Development Process Products

Each of the eight development phases produce specific deliverables. Some of those deliverables evolve into the product itself while other deliverables document the development project. Some deliverables fit into both categories. Examples of deliverables which evolve into the product itself include product descriptions and specifications, code used to lay out chips, and prototypes. The ICIDS groups the documentation deliverables into four documents which are used for phase reviews:

- A business plan
- A product specification
• A quality plan
• A launch plan

The documentation deliverables provide descriptions and measures of the Python development project. They grow and evolve through the project.

5.3.5 Product Development Project Metrics

ICI uses the three most common product development project metrics to measure project performance\(^{5,2}\): cycle time, quality, and cost (Montoya-Weiss and Calantine, 1994; Griffin and Page, 1993; Rosenau and Moran, 1993; Rosenthal, 1992). Cycle time is the dominant metric since ICI considers market windows to be relatively short and competition strong. This emphasis is consistent with changes in product development and competitiveness (Cooper and Kleinschmidt, 1994; Page, 1993; Merrills, 1991). Cycle time is measured from the beginning of Phase 1 to the Phase 3 review and from the Phase 3 review to Phase 7 review.

Quality is addressed passively through the ICI culture. ICI has a history of developing high quality products and a culture which considers flaws released to customers to be indicators of poor product development. External product quality (i.e. quality as experienced by customers) is counted in flaws discovered after Phase 7 (Launch) but is not formally measured or recorded. Quality internal to the development process is not measured.

Cost is measured with the expenses (primarily labor) charged to a product development team and functional groups. Costs of individual projects are not separated. Cost is the least important product development project metric at ICI. One informant said "We can spend whatever we want."

\(^{5,2}\) Product development project success is differentiated from new product success in that the former ends at the beginning of steady state production whereas the latter extends into the product life and typically includes metrics which reflect the product's financial performance (Page, 1993).
5.3.6 The Development Organization

ICI directly manages and is responsible for all the significant functions and groups which contribute to development projects. The development organization at ICI is a matrix structure with function-based departments and product development managers acting orthogonally (Mintzberg, 1979). The structure is similar to those used in other industries (Wheelwright and Clark, 1992) and estimated to function between Clark and Fujimoto's (1991b) lightweight and heavyweight product manager classifications. Different functional groups provide primary, secondary, and supportive functions in the different development phases. Table 5-1 shows the interfaces of development phases and functional groups for a single project.

5.4 The Python Development Project

Python: was a new product built from existing technologies which the company already understood. The product would be manufactured with a known, tested, and familiar manufacturing process. Therefore the product and not the manufacturing process was the focus of the Python project and is therefore the focus of the application of the Product Development Project Model. Python was expected to have a relatively short lifetime (a few years) compared to the division's traditional products (decades in some cases). Therefore getting development completed quickly was considered paramount to success. This emphasis on time was reinforced by the company history and the nature of the semiconductor industry. From a feedback perspective this implies that feedback loops relating schedule performance to project management decisions would be expected to be stronger than other project target feedback structures.
<table>
<thead>
<tr>
<th>Phase</th>
<th>Department</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Discover</td>
</tr>
<tr>
<td>2</td>
<td>Metrics</td>
</tr>
<tr>
<td>3</td>
<td>Prod. Definition</td>
</tr>
<tr>
<td>4</td>
<td>Design</td>
</tr>
<tr>
<td>5</td>
<td>Fab &amp; Assembly</td>
</tr>
<tr>
<td>6</td>
<td>Test &amp; Char</td>
</tr>
<tr>
<td>7</td>
<td>Launch</td>
</tr>
<tr>
<td>8</td>
<td>PD Review</td>
</tr>
</tbody>
</table>

**LEGEND**

- **P** - Primary participant in development phase
- **S** - Secondary participant in development phase
- **C** - Continuous support for primary and secondary participants
- **I** - Intermittent participation or support in development phase
The Python development team included representatives from all the functional groups listed in Table 5-1. The team is relatively small with one to three persons typically representing each function on the team. The average experience in computer chip development by the Python team is greater than five years. In addition, turnover at ICI is quite low so many portions of the Python development team had worked together on many development projects. The combined impacts of a familiar manufacturing process, industry-specific development experience, and extensive cooperative development experience eliminated or significantly reduced some of the learning curve and training effects on the Python project.

ICI has a history of delivering defect-free products to customers. This is based on having developed many redesigns of existing products and many products (typically memory chips) which are very simple compared to Python. Therefore sincere expectations that Python would attain a quality goal of no defects after launch was realistic from ICI's historical perspective. In practice this was not achieved. Two flaws requiring partial remasking and several flaws which were addressed with software drivers developed by the development team in parallel with the Python chip were found jointly with a major customer in the three months following Python's launch. Product development project cycle times at ICI were historically longer than originally planned and considered a problem. The use of very aggressive schedule goals had led to unusually high estimates of project durations and other artificial schedule adjustments. Costs also traditionally exceeded original estimates but were not considered a problem because it was believed that these overruns could be more than compensated for with a good quality and schedule performance.
5.5 Model Calibration

5.5.1 Model Structure and the Python Development Project

5.5.1.1 Overview

The structure of the Product Development Project Model and the Python project are similar. Table 3-1 identifies the model sectors in which the structure is based partially on field data. The descriptions of parameter estimates in this chapter identify the frequent use of observed data to identify, define, and estimate model parameters. Parameter dimensions were used to assure consistency among relationships. Partial and complete model configurations were tested under extreme conditions for reasonable behavior (Homer, 1983). These simulations and tests increase the confidence in the model's ability to simulate the Python development project (Forrester and Senge, 1980).

The eight development phases in ICI's ICIDS were modeled with four model phases in the Product Development Project Model. The selection of activities to be modeled was based primarily on the specifics of the Python project and data availability. More specifically no Phase 8 (project evaluation) was performed. Very little data was available concerning the phases dominated by the marketing functions (phases 1, 2, and 7). The remaining phases (3, 4, 5, and 6) focused on the development of the physical product. The names of some model phases were purposefully selected to be different than the ICIDS phase names to avoid confusion and as a reminder of the difference between the ICIDS development phases and the model phases. The model phases, their primary activities, and the basis for describing their tasks follow:

- **Product Definition model phase**: Prepare a full technical description of the product, its features and its functions. These descriptions took the form of
product specifications which grew and changed during the project. Specification items were chosen as the development task for this model phase.

- **Design model phase**: Translate the product's technical description into a description of the physical computer chip. Review product definition specifications for designability. The product of this work is software code known as RTL code which designers write and is subsequently used for layout. Lines of code could be used as a development task for this model phase, however that information is considered sensitive information. The RTL code for the Python chip is quite uniform in density when printed. Therefore inches of code could be safely substituted for lines of code as a measure of development product for this model phase.

- **Prototype Testing model phase**: Build test prototypes from the description of the physical chip and compare performance with product specifications. Correct specifications and add remaining functional specifications as required. This phase performs their work on specifications from the Product Definition model phase and prototypes which are directly related in scope to the tasks in the Product Definition and Design model phases. Therefore specification items and inches of RTL code were used as the basis for describing development tasks for this phase.

- **Reliability/Quality Control model phase**: Use the specifications tested for product functionality and the prototypes built from the description of the physical chip to test the product for large scale manufacturability. This phase performs their work on specifications from the Product Definition model phase and prototypes which are directly related in scope to the tasks in the Product Definition and Design model phases. Therefore inches of RTL code were used as the basis for describing development tasks for this phase.

The flow of work described above can be depicted with a Project Phase Network, as shown in Figure 5-2.
Figure 5-2: Project Phase Network for the Python Development Project

Each of the model phases which base their work on an upstream phase (e.g. Design is based on Product Definition) also checks the inherited work for errors and returns flawed tasks for correction\textsuperscript{5,3}. The only exception to this rule is the Prototype Testing phase which inherits specifications from the Product Definition model phase but corrects any specification errors discovered within the Prototype Testing phase instead of returning them for correction.

5.5.1.2 Model Boundary Test

Data was collected for a model boundary test (Forrester and Senge, 1980). The purpose of the test was to identify project features and relationships which are significant to project performance but were not included in the Product Development Project Model. The basic approach taken was to repeatedly draw descriptions of important project features and relationships from developers and managers in different ways to identify significant model structure.

\textsuperscript{5,3} The Product Development Project Model isolates the movement of tasks to the confines of a single model phase. The movement of flawed tasks between phases is modeled with information flows and not actual task flows. See chapter 3 for details. The traditional terminology will be used in this chapter for clarity.
Open-ended interviews were held with managers and developers on the Python development team. The interviews sought the answers to three questions from each informant:

- What are the important measures of project performance?
- What factors drive project performance?
- How are the most influential of those factors related?

The interviews generated both "hard" (easily quantifiable) and "soft" (often qualitative) project factors. These potential variable names were grouped into the categories shown in Table 5-2.
<table>
<thead>
<tr>
<th>Project Variable</th>
<th>Number of Times Identified</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>The Development Process</strong></td>
<td></td>
</tr>
<tr>
<td><strong>Information Flow</strong></td>
<td></td>
</tr>
<tr>
<td>Information sharing with market and customers</td>
<td>10</td>
</tr>
<tr>
<td>Quantity of information flow</td>
<td>8</td>
</tr>
<tr>
<td>Quality of information flow</td>
<td>6</td>
</tr>
<tr>
<td>Development Process Information Flow subtotal</td>
<td>24</td>
</tr>
<tr>
<td><strong>Development Process Structure</strong></td>
<td></td>
</tr>
<tr>
<td>Stage-Gate structure (ICIDS)</td>
<td>7</td>
</tr>
<tr>
<td>Degree of concurrency in process</td>
<td>3</td>
</tr>
<tr>
<td>Development Process Structure subtotal</td>
<td>10</td>
</tr>
<tr>
<td><strong>Changes</strong></td>
<td></td>
</tr>
<tr>
<td>General (role of, number of, etc.)</td>
<td>19</td>
</tr>
<tr>
<td>Quality Assurance</td>
<td>11</td>
</tr>
<tr>
<td>Rework</td>
<td>7</td>
</tr>
<tr>
<td>Start and Stop delays</td>
<td>1</td>
</tr>
<tr>
<td>Changes subtotal</td>
<td>38</td>
</tr>
<tr>
<td><strong>Development Process Infrastructure &amp; Support</strong></td>
<td></td>
</tr>
<tr>
<td>Tools available</td>
<td>6</td>
</tr>
<tr>
<td>Coordination of development process</td>
<td>5</td>
</tr>
<tr>
<td>Development Process Infrastructure &amp; Support subtotal</td>
<td>11</td>
</tr>
<tr>
<td><strong>Management of the Development Process</strong></td>
<td></td>
</tr>
<tr>
<td>Good product definition</td>
<td>9</td>
</tr>
<tr>
<td>Performance pressure</td>
<td>4</td>
</tr>
<tr>
<td>Systemic perspective of process by management</td>
<td>1</td>
</tr>
<tr>
<td>Management of Development Process subtotal</td>
<td>14</td>
</tr>
<tr>
<td>Development Process subtotal</td>
<td>97</td>
</tr>
<tr>
<td><strong>Resources</strong></td>
<td></td>
</tr>
<tr>
<td><strong>People</strong></td>
<td></td>
</tr>
<tr>
<td>Quantity available or used</td>
<td>25</td>
</tr>
<tr>
<td>Capabilities, knowledge, or experience</td>
<td>12</td>
</tr>
<tr>
<td>Delays in getting &quot;up to speed&quot; about the project</td>
<td>4</td>
</tr>
<tr>
<td>Project personnel turnover rate</td>
<td>1</td>
</tr>
<tr>
<td>Delays in getting the right people</td>
<td>1</td>
</tr>
<tr>
<td>People Resources subtotal</td>
<td>43</td>
</tr>
</tbody>
</table>

Table 5-2: Project Variable Types Generated by Interviews (partial)
<table>
<thead>
<tr>
<th>Project Variable</th>
<th>Number of Times Identified</th>
</tr>
</thead>
<tbody>
<tr>
<td>Non-human resources</td>
<td></td>
</tr>
<tr>
<td>Quantity available or used</td>
<td>15</td>
</tr>
<tr>
<td>Capabilities</td>
<td>4</td>
</tr>
<tr>
<td>Delay due to unavailability</td>
<td>2</td>
</tr>
<tr>
<td><strong>Non-human Resources subtotal</strong></td>
<td>21</td>
</tr>
<tr>
<td>Other</td>
<td></td>
</tr>
<tr>
<td>Unspecified resource type availability</td>
<td>8</td>
</tr>
<tr>
<td>Technology used in development</td>
<td>4</td>
</tr>
<tr>
<td><strong>Resources: Other subtotal</strong></td>
<td>12</td>
</tr>
<tr>
<td><strong>Resources subtotal</strong></td>
<td><strong>76</strong></td>
</tr>
<tr>
<td><strong>The Development Organization</strong></td>
<td></td>
</tr>
<tr>
<td>Development Team</td>
<td></td>
</tr>
<tr>
<td>Vision and direction of development team</td>
<td>9</td>
</tr>
<tr>
<td>Design of development team</td>
<td>7</td>
</tr>
<tr>
<td>Rewards for development team</td>
<td>7</td>
</tr>
<tr>
<td>Cohesion and morale of team</td>
<td>7</td>
</tr>
<tr>
<td>Team meetings</td>
<td>3</td>
</tr>
<tr>
<td>Systemic perspective of process by team</td>
<td>1</td>
</tr>
<tr>
<td><strong>Development Team subtotal</strong></td>
<td><strong>34</strong></td>
</tr>
<tr>
<td>Management and Leadership</td>
<td></td>
</tr>
<tr>
<td>Support of team and project</td>
<td>6</td>
</tr>
<tr>
<td>Style and making of decisions</td>
<td>2</td>
</tr>
<tr>
<td>Delays in making decisions</td>
<td>1</td>
</tr>
<tr>
<td><strong>Management and Leadership subtotal</strong></td>
<td><strong>9</strong></td>
</tr>
<tr>
<td><strong>Development Organization subtotal</strong></td>
<td><strong>43</strong></td>
</tr>
<tr>
<td><strong>The Development Environment</strong></td>
<td></td>
</tr>
<tr>
<td>The business cycle</td>
<td>1</td>
</tr>
<tr>
<td>The needs of the market</td>
<td>1</td>
</tr>
<tr>
<td><strong>Development Environment subtotal</strong></td>
<td><strong>2</strong></td>
</tr>
<tr>
<td><strong>TOTAL VARIABLE IDENTIFICATIONS</strong></td>
<td><strong>218</strong></td>
</tr>
</tbody>
</table>

Table 5-2: Project Variable Types Generated by Interviews
The interviews also identified the relationships which the developers and managers considered most influential on project performance. To facilitate the elicitation of the knowledge about the project structure from the informants I also led a workshop at ICI with approximately a dozen developers. The management, strategic marketing, launch marketing, product definition, design, and prototype testing functions were represented. The workshop sought to identify the primary feedback loops considered important in describing a development project at ICI. After an introduction to the scope and purpose of the meeting the workshop developed a list of measures of a project. The potential variable names fit into the categories in Table 5-2 above. The meanings of these project parameter names were clarified and roughly prioritized by their influence on project performance. These parameters became variables in the causal feedback loops. The majority of the workshop was spent constructing feedback loops by identifying causal relationships. The development team had difficulty focusing on the single-project level of aggregation, preferring to describe multiple project political issues controlled by upper level managers or developing increasingly detailed feedback structures in a relatively narrow band of topics centering around resources. Introducing additional parameters from the original list facilitated the investigation of project-level structures. The primary project-level feedback structures identified at the model boundary workshop are shown in Figure 5-3.
The lack of negative feedback loops identified by the managers and developers was noticed during the model boundary workshop. Despite suggestions and encouragement to identify negative feedback loops the managers and developers identified only positive feedback loops at the project level.

The model parameters and feedback structures identified by the managers and developers are included in the Product Development Project Model through its assumptions and structure. Many features are represented in the same form identified by the developers.
(e.g. stage-gate structure, degree of concurrence, resource quantity, rework, coordination). Others are reflected in different model parameters. For example "good product definition" is modeled by the number of errors released by the Product Definition model phase.

5.5.1.3 Customization of the Product Development Project Model Structure to Reflect the Python Development Project

The observed structure of the chip development process revealed a significant discrepancy between the process as practiced at ICI and the Product Development Project Model's representation of that process as described in chapter 3. The specific difference is in the timing of the release of design tasks to downstream phases. As described in chapter 3 the Model releases tasks continuously as they are checked for flaws and found to be flawless. However in practice the Design activity releases their completed and checked tasks differently. The Design activity generates RTL code which is released to a different group for the layout of the chip's layers. These layouts are made into masks which are subsequently used to make the chip layers. Mask making is quite expensive, often costing tens of thousands of dollars for a complete set of masks. To minimize the cost of remasking the number of times that the chip layout is changed and new masks are made is minimized. This is facilitated by holding RTL code in the design phase until it is considered complete and correct and then releasing the aggregated set of tasks. A different structure is required in the Model to reflect this aggregation, holding and aggregate release of RTL code. In general an additional stock is needed between the Completed but not Checked stock and the Tasks Released stock to collect and hold tasks until all the tasks believed to be needed are completed and considered free of flaws. This added stock and the changes required to integrate it into the existing model structure are shown in Figure 5-4.
Figure 5-4: The Additional Stock Required to Model the Aggregate Release of Tasks from a Phase

The new stock named "Hold for Release" requires the additional structure shown in Figure 5-5 to simulate the decision process of the developers for releasing tasks. The objective of the added structure is to reflect both the accumulation of tasks until the work is considered to be complete by the designers and the subsequent release of all tasks being held. Two circumstances must be modeled with this structure: 1) the initial release of nearly all the tasks and 2) the subsequent accumulation of corrected tasks until the designers believe all flawed tasks have been found and corrected and the resulting re-release of this typically smaller collection of tasks. A direct comparison of the number of tasks in the Hold for Release stock and the total number of tasks in the phase (the Task List) can model the first condition but not the second. The solution taken is to model the
release criteria with the variability of the stock of held tasks, i.e. the stability of the Hold for Release stock’s average value. This is based on the reasoning that when the size of this stock remains stable for a sufficient period the designers would believe that they had completed all the tasks, discovered and corrected all the flaws, and received and corrected all the flawed tasks from downstream phases, thereby justifying the release of the tasks to the layout activity.

Figure 5-5: Additional Model Structure Representing the Aggregation and Holding of Development Tasks before Release
The following relationships and equations model the aggregate-release structure. The Hold for Release stock accumulates the flows of tasks from the Completed but not Checked stock and to the Tasks Released stock.

\[
\text{Hold\_for\_Release(Phase)} = \text{Hold\_for\_Release(Phase)} + dt \times (\text{Tasks\_to\_Release\_Hold(Phase)} - \text{Release\_Tasks\_from\_Hold(Phase)})
\]

The Tasks to Release Hold flow is the number of tasks checked and considered unflawed. It is identical to the Release Tasks flow described in chapter 3.

\[
\text{Tasks\_to\_Release\_Hold(Phase)} = (\text{QA\_inspection\_rate(Phase)} - \text{RW\_due\_to\_InPhase\_QA(Phase)} - \text{RW\_due\_to\_Corrupted\_tasks(Phase)})
\]

The Release Tasks from Hold flow dumps all the tasks being held plus the tasks released to hold at the time of the release into the Tasks Released stock in a single time increment when the Release Trigger is "pulled" (equal to 1 and not 0).

\[
\text{Release\_Tasks\_from\_Hold(Phase)} = \text{Release\_Trigger(Phase)} \times ((\text{Hold\_for\_Release(Phase)})dt) + \text{Tasks\_to\_Release\_Hold(Phase)}
\]

The rest of the structure controls when the Release Trigger is "pulled". The Release Trigger is actuated only when the number of tasks held is more stable than a criteria named the Release Trigger Sensitivity.

\[
\text{Release\_Trigger(Phase)} = \text{FIFGE}(0,1,\text{MAX(Release\_Hold\_Avg\_Stability(Phase)}, (-1)\times\text{Release\_Hold\_Avg\_Stability(Phase)})\times\text{Release\_Trigger\_Sensitivity(Phase)})
\]

The Release Hold Average Stability parameter describes how quickly the designers assume that their work is complete. The stability of the Hold for Release stock is defined as the difference between the number of tasks being held and the average of that number. A lower difference reflects less change in the stock and more likelihood that all the work is completed.

\[
\text{Release\_Hold\_Avg\_Stability(Phase)} = \text{Release\_Hold\_Avg(Phase)} - \text{Hold\_for\_Release(Phase)}
\]
The Release Hold Average moves toward the Hold for Release level at a rate equal to their difference smoothed by the Release Hold Average Time. This time constant depicts the time needed for the designers to recognize, report, and internalize the completion of tasks into the stock of held tasks.

\[
\text{Release Hold Avg(Phase)} = \text{Release Hold Avg(Phase)} + \frac{\text{dt}}{} \times (\text{Change in Release Hold Avg(Phase)} - \text{Empty Release Hold Avg(Phase)})
\]

\[
\text{Change in Release Hold Avg(Phase)} = \left( \frac{\text{Hold for Release(Phase)}}{\text{Tasks to Release Hold(Phase) - Release Hold Avg(Phase)}} \right) / \text{Release Hold Avg Time(Phase)}
\]

Finally, the Empty Release Hold Average flow resets the Release Hold Average stock to zero when tasks are released so that the aggregation and release process can be repeated.

\[
\text{Empty Release Hold Avg(Phase)} = \text{FIFZE(0, Release Hold Avg(Phase))/dt, Release Trigger(Phase)}
\]

The new structure was spliced into the Development Tasks model structure parallel to the existing Release Tasks flow from the Completed, not Checked stock to the Tasks Released stock. A similar and directly analogous structure representing the aggregation, holding and aggregate release of errors was added to the Internal Errors sector parallel to the Release Errors flow. A switch is used to direct the flow of tasks through the continuous-release or aggregated-release structure.

### 5.5.2 Parameter Estimation

Parameters were estimated for two model configurations: a one phase model of the Design phase and the four phase model shown in Figure 5-1. The data described in section 5.2 was supplemented by previous research reported in the literature.
5.5.2.1 One Phase Model Parameter Estimates

This section describes the system description parameter estimates used to calibrate the one phase model to the design phase of the Python development project. A few observations about the design activity of the Python project help explain many of the parameter settings. The RTL code was generated, tested for errors, and corrected by only two people. For much of the design phase only one of these two men was working on the project at a time. Both men are experienced chip designers and had worked together on several projects before the Python project. They therefore were relatively efficient in their communication and interactions compared to a new or larger team of developers. One important result of these conditions is that several of the delays which are important in larger development contexts did not influence the design of the Python chip as significantly as they would a larger project or development team. For example, the time required to collect, aggregate, synthesize and report actual productivity to developers for use in developing productivity expectations may be measured in weeks for a medium to large group with a formal data reporting process. But this delay becomes very small for a single developer who "reports" his or her own information only to themselves. This is particularly evident in the parameters which describe productivity because of the important role of reporting and perceptions in productivity estimates.

The division's tradition of error-free product development and quality improvement programs generated sincere expectations that Python would attain a quality goal of no defects. Regardless of the accuracy of this expectation, this goal was realistic from the division's historical perspective. One impact of the emphases on speed and quality on both the Python project and the calibration of the single phase model is the disconnection of cost factors from the management of the Python project. This is confirmed by several interviews with developers and managers and reflected in the parameter settings below.

---

5.4 The other type of parameter is model control parameters. See Appendix 5.1 for a complete listing of parameter values for calibration.
Process Parameters
Basework Minimum Task duration (Design) = 2 (hours per inch of code)
Rework Minimum Task Duration (Design) = 1 (hours per inch of code)
Quality Assurance Minimum Task Duration (Design) = 0.75 (hours per inch of code)

These are measures of the relative times required to perform Basework, Quality Assurance, and Rework. Abdel-Hamid and Madnick (1991) estimated the values for parameters that have a similar meanings to the Basework Minimum Task duration to be between 40 and 60 lines of code per man-day (pg. 143 and 155). The Python code density averages 6 lines per inch of code (by inspection of the code printout). Since a task is assumed to be developing an inch of code Abdel-Hamid and Madnick's estimates would therefore be 6.7 (=40/6) and 10 (=60/6) lines of code per man-day or 0.3 (=6.7/24) and 0.4 (=10/24) inches per hour or 3.6 and 2.4 hours per task. Based upon differences in effects included in the variable and estimation methods between Abdel-Hamid and Madnick and this model, the purely process values used for this model are estimated to be slightly lower. Therefore the value of 2 for the Basework Minimum Task Duration is reasonable. Values for the Rework and Quality Assurance durations are estimates of the relative time required compared to the Basework value. Rework is estimated to take half the time due to the availability of knowledge about the probable location of the error being corrected and therefore potential to focus efforts to only a portion of a task (i.e. less than all 6 lines of code in the inch of code which is flawed). Quality Assurance is considered to potentially progress at an even faster pace due to the ability to test large pieces of code simultaneously and the use of standardized error detection methods.

Internal Precedence Relationship: (dimensionless)
Values of Percent Available for Completion for the values of Percent Completed and Released = 0.1, 0.2, ...0.9, 1.0 are: 0.01/0.21/0.31/.41/0.51/0.61/0.71/0.81/0.91/1.00/1.00

This plot stays above the 45 degree line to prevent an internal "death grip" in which the phase can not produce the work required to proceed. Figure 5-6 shows the estimates of this parameter generated by two design engineers and a design manager at ICI. The plot used for the calibration includes the slow initial startup and the relatively stable slope toward release of all tasks at 90% completed and released. The highest estimate was
produced by the design manager and therefore given less weight than those generated by the design engineers in estimating this parameter. The lower of the two designer estimates was judged to reflect the actual relationship based on conversations with the designers.

![Diagram](image)

**Figure 5-6: Design Phase Internal Precedence Relationship for Calibration**

Release_Trigger_Sensitivity(Design)=0.1 (dimensionless)

This parameter describes how long the team waits for the number of tasks completed and held for release to remain stable before believing that they are all done and releasing the code to layout. The value 0.1 indicates that the tasks completed and held for release must differ from the average of that value by 0.1 tasks. This difference allows the release with a very few non-critical tasks remaining in rework.
Release_Hold_Avg_Time(Design)=1.05 (weeks)
As the code nears completion the designers watch very closely and are pressured to release code to layout as quickly as possible. Therefore they incorporate the addition of more code to the "done" stocks very quickly.

**Scope Parameters**
Task_List(Design)=445 (tasks)
The total number of tasks in the design phase is 504. Fifty nine of those tasks occur early and are separated from the other 445 tasks by several weeks of zero design activity. Based upon interviews with the designers, the stoppage was due totally to the product definition phase not being complete enough to proceed farther with design (i.e. a pure External Precedence Relationship constraint). This interruption should not and can not be modeled in a one-phase test and therefore the 59 tasks were deleted for this calibration. The remaining 445 tasks are the basis for the calibration. The planned scope did not change during the design phase.

**Target Parameters**
Schedule
Initial_Prop_Deadline=25 (weeks from phase start)
The duration goal for phases four to six goal is 64 weeks for complex products such as the Python chip (ICI internal document). Estimates of non-design activities are (from interviews):

Four weeks for lay out + two weeks for maskmaking + eight weeks for wafer fabrication + seventeen weeks for testing = 31 weeks total for non-design activities
64 total weeks - 31 nondesign activity weeks = 33 weeks for design.

I adjusted the 31 week estimate down based on the Phase 3 review report which estimated that the Phase 4 (design) duration would be 8-10 weeks. I used an estimate of 25 weeks for the design phase.
Time to Average Expected Completion Time(Design)\(=1\) (weeks)
Revised estimates of phase durations are assumed to be incorporated into new estimates of completion dates within a week.

Resistance to Schedule Slip\(=2\) (dimensionless)
Based upon interviews and ethnographic data the Python organization is in a constant state of schedule pressure and is therefore able to resist relatively high levels of schedule pressure before reacting (slipping the deadline). This value indicates that deadlines will not slip until the estimated time to complete the phase is twice the time available.

Quality
Quality\_Goal\_Adjust\_Time(Design)\(=24\) (weeks)
High quality as measured by few released errors has a long history based in the company's history of doing simple products and redesigns which could be developed and released without errors. Two interviews (with the project manager of Python and the product definition and development process improvement engineer) give anecdotal data supporting a very strong quality ethic. The project manager described ICI as being "quality obsessed" and only beginning to learn that tradeoffs between quality and other performance measures are available. The value is approximately the original phase duration estimate of 25 weeks, implying that quality can not slip during a single phase.

Project\_Quality\_Goal=1.00 (dimensionless)
Initial\_Quality\_Goal(Phase)=1.00 (dimensionless)
High quality as measured by few errors has a long history based in the company's history of doing simple products and redesigns which could be developed and released without errors. Two interviews give anecdotal data supporting a very strong expectation of management and developers of products which are released to customers with no errors. The ethic is assumed to translate to the design phase for this calibration.
Basic_prob_flawed_Task(Design)=0.85 (dimensionless)
The Python code was written in 17 blocks. Each block provided a specific function and therefore can be seen as a logical location of coding errors. An estimate based upon discussion with a designer and a relatively high number of code changes (111 changes to the 17 functional packets of code). Because this is a single phase model this probability includes both inherited errors and internally generated errors. Therefore it is expected to be somewhat higher than an estimate for a single phase within a multiple phase model. See multiple phase model parameter estimates below for more detail.

Complexity(Design)=10 (dimensionless)
This value is significant only relative to the reference complexity of 100. This estimate is based upon the use of established and standard manufacturing processes and known technologies to build Python.

Cost
Budget_Switch=0 (dimensionless)
Several interviews (e.g. Development Project Manager) indicate that development costs have no impact on the operation of development process, including on the Python project. This setting disconnects the cost feedback loops.

Resource Parameters
Gross Labor
Initial_Headcount(Design)=0.50 (developers)
Release of a phase typically includes the reassignment of a developer to the project on a part time basis until work load increases. This is an estimate of the minimum realistic time assignment for the progression of work, including all the required administrative time.
Max_Headcount(Design)=2 (developers)
This estimate is based on interviews with developers about the likelihood of getting more than two full time, rested, experienced equivalent persons (zero likelihood). Because of the effects of inexperience, fatigue, and multiple assignments filling a maximum headcount of two persons requires more than two actual people.

Headcount_Adjustment_Time(Design)=8 (weeks)
This estimate is based on the following, which is paraphrased from the interview with development project manager: "You pretty much have to work with what you've got." [Increasing headcount is very difficult.]. ICI's total resources limit forces "stealing" and luring of developers among development projects. The time to adjust headcount (in weeks) includes time to identify the person, negotiate the transfer with the developer and their superior, transition time from one project to another, and time to get "up to speed" on the new project. The following supports this estimate and is from an interview about the design phase: The typical time to actually shift someone is approximately 3-4 weeks. If a critical need is experienced it can be done in 2-3 weeks. The time required to get "up to speed" is very dependent on whether the person has worked before with the part he or she will be working on. This time is short if they have experience but 4-8 weeks if they do not have experience. I used $4 + 4 = 8$ weeks.

HdctJumpStartTime(Design)=11 (weeks from start)
This is the starting time of the jump in the headcount. The Design phase experienced an exogenous change in the headcount when both designers working on the project were reassigned for 3 weeks and then returned to the Python design phase. This parameter and the next two parameters describe this change. This estimate is based on data from the interview with the process engineer based about his discussion with designer.

HdctJumpStopTime(Design)=14 (weeks from start)
This is the stopping time of the jump in the headcount. This estimate is based on the interview with the process engineer based his discussion with designer.

Max_Workweek(Design)=140 (hours per week)
Date and time stamps on e-mail messages document and developers relate in interviews that the developers are willing to work for short periods at rates significantly more than 80 hours per week.

Normal_Workweek(Design)=40 (hours per week)
This is based on the assumptions of a five day week and eight hour day.

Wrkwk_Avg_Time(Design)=4 (weeks)
This is the time used to average the workweek. This is an estimate by the modeler of the time for fatigue effects to be generated from extended weeks of overtime and is based on developer estimates.

**Labor Allocation**

BW_Priority(Design)=3 (dimensionless)
RW_Priority(Design)=1 (dimensionless)
QA_Priority(Design)=1 (dimensionless)

These are estimates of the relative importance given to these activities based upon ethnographic data concerning importance of schedules and the assumption that quality assurance and rework are partially "built into" Basework activities via ICI's several ongoing improvement programs.

BW_Labor_Delay(Design)=1.5 (weeks)
This parameter is the delay between the generation of demand for basework and the actual application of labor for basework. This estimate is based upon interviews concerning the difficulty and delay in making labor changes within projects.
RW_Labor_Delay(Design)=1 (weeks)
This parameter is the delay between the generation of demand for rework by the increase in the Known Rework stock and the actual application of labor for rework. This estimate is based on the assumption that the decision to spend time on rework is internal to the development team and indistinguishable from Basework to those not doing the activity and therefore can be made with minimal delay.

QA_Labor_Delay(Design)=7.75 (weeks)
This parameter is the delay between the generation of demand for Quality Assurance by the increase in the Completed Tasks stock and the actual application of labor for Quality Assurance. This parameter can be estimated from reference mode data from the design phase of the Python project by comparing the horizontal distance between the increase in the Completed Tasks stock and the Quality Assurance rate. Unfortunately direct and isolated measures of the size of the Completed Tasks stock are unavailable (available data combines this stock with the Held for Release stock). But in the initial portion of the design phase the flows out of the Completed Tasks stock to Known Rework and Hold for Release can be safely assumed to be negligible. Therefore the size of the Completed Task stock can be estimated by the integration of its inflow from Basework. This value is called the Cumulative Basework and is plotted with the Error Checking (QA) rate below. The delay is estimated as:

Week 13 (QA rate starts) - Week 5 (demand for QA builds) = 8 week delay

Since this is a one phase model estimate it includes (i.e. internalizes) the effects of inheriting specifications from the Product Definition phase and is expected to be higher than the same parameter in a multiple phase model calibration.
Experience
Exper_Assim_Time(Design)=1 (weeks)
This time is an estimate based on a very small (less than 2 persons) development team of experienced and efficient communicating designers restrains delay in applying experience.

Productivity
Ref_BW_Prdctvty(Design)=2 (tasks per person per hour)
Ref_RW_Prdctvty(Design)=1 (tasks per person per hour)
Ref_QA_Prdctvty(Design)=1.75 (tasks per person per hour)
These values represent the relative productivities of the three pure (no impacts of experience, schedule, etc.) development activities. In accordance with the assumption that a development task represents approximately an hour of development work, these estimates remain within an order of magnitude of unity (i.e. 0-10). Generating new code
on blank pages (Basework) is assumed to be completed twice as quickly as identifying
the nature of an error after the existence of an error is identified and correcting the error
(Rework). Therefore Basework productivity is estimated as 2 tasks per person per hour
and Rework productivity is estimated as 1 task per person per hour. Finding errors (QA)
is estimated to progress faster than correcting those errors (Rework) because of the
availability and use of design code testing software by the Python developers but slower
than Basework.

BW_Prdctvty_Avg_Time(Design)=1 (weeks)
RW_Prdctvty_Avg_Time(Design)=1 (weeks)
QA_Prdctvty_Avg_Time(Design)=1 (weeks)
Adjust_Expect_BW_Prdctvty_Time(Design)=1 (weeks)
Change_Expected_QA_Prdctvty_Time(Design)=1 (weeks)
BW_Prdctvty_Influences_Time(Design)=1 (weeks)
Ch_Expect_Coord_Prdctvty_time(Design)=1 (weeks)
QA_Prdctvty_Report_Time(Design)=1 (weeks)
Report_BW_Prdctvty_Time(Design)=1 (weeks)

These times slow and smooth the averaging, reporting, and estimating of productivities
by the developers. As discussed previously the number of developers is very small, they
have large amounts of previous development experience and had worked together
previously. As importantly the development process and organization includes no formal
process for collecting, aggregating and reporting productivity. The processes which this
delay slows and smoothes are carried out by the two developers in a very small, very
informal, typically intuitive and qualitative and probably sometimes unconscious way.
Therefore the delay is considered to be minimal.

Wt_to_Current_BW_Prdctvty(Design)=1.00 (dimensionless)
The Python product was new for the division and designers. Based upon interviews, the
designers expected their productivity to be significantly different than their productivity
on their historically simpler products. Therefore they disregarded their historical
productivity estimates and based their expectations on their experience with the Python
project.
5.5.2.2. Multiple Phase Model Parameter Estimates

Four sets of system description parameter values are required to calibrate the multiple
phase model shown in Figure 5-2. The values for the design phase are described above.
The values for the Product Definition, Prototype Testing and Reliability/Quality control
phases which differ from the design phase values are shown in Table 5-3.

<table>
<thead>
<tr>
<th>Parameter Name</th>
<th>Product Definition</th>
<th>Design</th>
<th>Prototype Testing</th>
<th>Reliability/Quality Control</th>
</tr>
</thead>
<tbody>
<tr>
<td>Task List</td>
<td>466</td>
<td>1219</td>
<td>1219</td>
<td>1219</td>
</tr>
<tr>
<td>Basic Prob Flawed*</td>
<td>0.5</td>
<td>0.3</td>
<td>0.05</td>
<td>0.05</td>
</tr>
<tr>
<td>BW Min Task Duration</td>
<td>5</td>
<td>2</td>
<td>6</td>
<td>2</td>
</tr>
<tr>
<td>QA Min Task Duration*</td>
<td>2</td>
<td>2</td>
<td>0.5</td>
<td>1</td>
</tr>
<tr>
<td>RW Min Task Duration*</td>
<td>3</td>
<td>0.5</td>
<td>0.5</td>
<td>1</td>
</tr>
<tr>
<td>QA Labor Delay*</td>
<td>12</td>
<td>3</td>
<td>0.5</td>
<td>3</td>
</tr>
<tr>
<td>BW Priority</td>
<td>3</td>
<td>3</td>
<td>5</td>
<td>3</td>
</tr>
<tr>
<td>QA Priority</td>
<td>1</td>
<td>2</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>RW Priority</td>
<td>1</td>
<td>2</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>Headcount Adjust Time*</td>
<td>12</td>
<td>8</td>
<td>8</td>
<td>8</td>
</tr>
<tr>
<td>Maximum Headcount*</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>Int. Precedence Rel*</td>
<td></td>
<td></td>
<td></td>
<td>See description below</td>
</tr>
<tr>
<td>Ext. Precedence Rel.s*</td>
<td></td>
<td></td>
<td></td>
<td>See description below</td>
</tr>
</tbody>
</table>

Table 5-3: System Description Parameter Values
for Multiple Phase Model Calibration

* - Identified in chapter 3 as a high leverage parameter.

Task List(Product Definition) = 466 (tasks)

This is the number of specification items developed by the product definition activity. It
was determined by a count of specifications using an estimate of a large paragraph of
product description or line of values in a table as a basis for task size.
Task List (Design, Prototype Testing, Reliability/Quality Control) = 1219 (tasks)
This is the estimated number of tasks required to describe the product requirements and physical characteristics. It was determined by a count of specifications and inches of RTL code.

Basic Prob Flawed (Product Definition) = 0.5 (dimensionless)
This parameter is expected to decrease in succeeding development phases as the product requirements and description evolves. This estimate is based on the relative difficulty of performing the tasks such that it will not require iteration. The 466 specification items experienced a total of 554 changes or deletions internally or in subsequent phases. However many of those changes were due to rework of the same specification items. The data shows this to be due to an extended discussion of what specifications the product should meet and the resulting repeated switching of product specifications for a relatively small set of specification items (an acceptable set of temperature ranges). These repeated switches during rework should not be considered in this estimate. The estimated total changes to original specification items is half the specification task list.

Basic Probability of a Flawed Task (Design) = 0.3 (dimensionless)
This parameter can be roughly estimated by decreasing the single phase estimate of 0.85 by the probability used for the Product Definition phase.

Basic Probability of a Flawed Task (Prototype Testing, Reliability/Quality Control) = 0.05, 0.05 (dimensionless)
These activities use self generated test programs or known test equipment. Errors in the test program are typically fixed in much less than an hour (interview with tester) and therefore assumed to be included in the basework duration estimate (see below). Reliability/Quality Control testing errors require quick adjustment to testing machines (interview with tester) and are assumed to be minimal. Tests relatively rarely give erroneous product quality results, estimated to be 5%. 
Basework Minimum Task Duration(Product Definition) = 5 (hours per task)
These are inherently long and difficult tasks. They are some of the most important
decisions about the product and are based on the least available information. They are
regularly rethought and revised within a single completion of the task for internal
acceptability (interview with strategic marketer). Estimated to be 5 hours.

Basework Minimum Task Duration(Design) = 2 (hours per task)
Abdel-Hamid and Madnick (1991) estimated the values for parameters that have a similar
meanings to the Basework Minimum Task duration for the writing of computer code to
be between 40 and 60 lines of code per man-day (pg. 143 and 155). The Python code
density averages 6 lines per inch of code (by inspection of the code printout). Since a task
is assumed to be developing an inch of code Abdel-Hamid and Madnick's estimates
would therefore be 6.7 (=40/6) and 10 (=60/6) lines of code per man-day or 0.3 (=6.7/24)
and 0.4 (=10/24) inches per hour or 3.6 and 2.4 hours per task. Based upon differences in
effects included in the variable and estimation methods between Abdel-Hamid and
Madnick and this model, the purely process values used for this model are estimated to be
slightly lower. Therefore the value of 2 hours for the Basework Minimum Task Duration
is reasonable.

Basework Minimum Task Duration(Prototype Testing) = 6 (hours per task)
This activity includes writing and debugging the test program and in-process test program
debugging. It is therefore estimated to be longer than would be the case with a defect-free
test program. It is estimated to be 6 hours.

Basework Minimum Task Duration(Reliability/Quality Control) = 2 (hours per task)
This activity uses known and familiar testing machines and is estimated to be 2 hours.
Quality Assurance Minimum Task Duration(Product Definition) = 2 (hours per task)
The Quality Assurance duration estimate is a relative time required compared to the Basework value. Most flaws in this activity are decisions that need revision and not mistakes. This requires less time than doing the work the first time (basework). It is estimated to be 2 hours.

Quality Assurance Minimum Task Duration(Design) = 2 (hours per task)
The Quality Assurance duration estimate is a relative time required compared to the Basework value. This estimate is longer than for the single phase model because the model phase now includes the checking of product specifications.

Quality Assurance Minimum Task Duration(Prototype Testing) = 0.5 (hours per task)
The Quality Assurance duration estimate is a relative time required compared to the Basework value. Finding errors in the testing method occurs relatively quickly. It is important to remember that this development activity is not finding product errors (the fundamental activity of this phase), but is finding errors in the testing of the product that exceed the in-process debugging of the test program discussed earlier.

Quality Assurance Minimum Task Duration(Reliability/Quality Control) = 1 (hours per task)
The Quality Assurance duration estimate is a relative time required compared to the Basework value. Finding errors in the testing method occurs relatively quickly. It is important to remember that this development activity is not finding product errors (the fundamental activity of this phase), but is finding errors in the testing of the product that exceed the in-process debugging of the test program discussed earlier.
Rework Minimum Task Duration (Product Definition) = 3 (hours per task)
The Rework duration estimate is a relative time required compared to the Basework value. The nature of this product development activity makes the correction of errors and revision of product definition products relatively slow.

Rework Minimum Task Duration (Design, Prototype Testing, Reliability/Quality Control) = 0.5, 0.5, 0.5 (hours per task)
The Rework duration estimate is a relative time required compared to the Basework value. Rework of code is estimated to take half of an hour on average due to the availability of knowledge about the probable location of the error being corrected and therefore the potential to focus efforts to only a portion of a task. This work includes finding errors in checking the specifications, which can occur quite quickly. This estimate is therefore smaller than in the one phase model.

Quality Assurance Labor Delay (Product Definition) = 12 (weeks)
This parameter is the delay between the generation of demand for Quality Assurance by the increase in the Completed Tasks stock and the actual application of labor for Quality Assurance. The ambiguity of projects in their early phases and the requirements of this activity and the availability of strategic marketing personnel caused this delay to be long.

Quality Assurance Labor Delay (Design, Prototype Testing, Reliability/Quality Control) = 3, 0.5, 3 (weeks)
This parameter is the delay between the generation of demand for Quality Assurance by the increase in the Completed Tasks stock and the actual application of labor for Quality Assurance. These are estimates.

Basework, Quality Assurance, and Rework Priorities (Product Definition): Basework = 3, Quality Assurance = 1, Rework = 1 (dimensionless)
These represent the relative importance of basework, quality assurance, and rework in a phase. Estimates based on interviews with developers in this phase. Basework is higher than the other priorities because of the importance of cycle time to project success.
Basework, Quality Assurance, and Rework Priorities (Design): Basework=3, Quality Assurance=2, Rework =2 (dimensionless)
These represent the relative importance of basework, quality assurance, and rework in a phase. Estimates based on interviews with developers in this phase. Basework is higher than the other priorities because of the importance of cycle time to project success. Some improvement and quality programs have focused on the design activity and therefore Quality Assurance and Rework priorities are higher than in other phases.

Basework, Quality Assurance, and Rework Priorities (Prototype Testing): Basework=5, Quality Assurance=1, Rework =1 (dimensionless)
These represent the relative importance of basework, quality assurance, and rework in a phase. Estimates based on interviews with developers in this phase. Basework is higher than the other priorities because of the importance of cycle time to project success. Completing gains importance as the project nears the end (independent of schedule pressure effects), thereby increasing the relative importance of basework.

Basework, Quality Assurance, and Rework Priorities (Reliability/Quality Control): Basework=3, Quality Assurance=1, Rework =1 (dimensionless)
These represent the relative importance of basework, quality assurance, and rework in a phase. Estimates based on interviews with developers in this phase. Basework is higher than the other priorities because of the importance of cycle time to project success.

Headcount Adjust Time(Product Definition, Design, Prototype Testing, Reliability/Quality Control) = 12, 8, 8, 8 (weeks)
Additional marketing personnel were usually unavailable for the Product Definition phase. These estimates are based on the interview with Development Project Manager who said "You pretty much have to work with what you've got." [Increasing headcount is very difficult.]. Total resources cap forces "stealing" and luring people off of other projects. The time to adjust headcount (in weeks) includes time to identify the person, transition time from one project to another, and time to get "up to speed" on the new project. For the design phase the typical shift time is approximately 3-4 weeks. If a
critical need arises it can be 2-3 weeks. The "up to speed" time is very dependent upon whether the person has worked before with the part he or she will be working on. The time is short if they have experience but 4-8 weeks if do not have experience.

Maximum Headcount(all phases) = 2 (developers)
This is the same as for the single phase model. This is an estimate of the full time rested experienced developers. It could therefore requires several actual persons to provide the labor to meet this limit.

Precedence Relationship Notation: Values are the eleven sequential parameter values for the input values of 0.0, 0.1, 0.2...0.8, 0.9, 1.0. Linear interpolation provides the values between these points.

\[
\text{Internal Precedence Relationship(Product Definition)} = \\
0.01/0.15/0.3/0.60/0.80/0.90/0.95/1.0/1.0/1.0/1.0
\]
This plot stays above the 45 degree line to prevent an internal "death grip" in which the phase can not produce the work required to proceed. The curve initially stays close to the 45 degree line, indicating the strong interconnection of the very early product definition decisions. When approximately 10% of the product definition tasks are made (overall architecture) more tasks become available quickly as individual subsystems are described. The curve approaches 100% available slowly as product definition decisions are integrated into a consistent set of specifications. This estimate is based on four estimates by a strategic marketing developer who works in product definition (upstream developer), two product architects (in-phase developer), a design manager (downstream development manager).

\[
\text{Internal Precedence Relationship(Design)} = \\
0.01/0.15/0.40/0.5/0.65/0.75/0.85/0.95/0.97/1.0/1.0
\]
This plot stays above the 45 degree line to prevent an internal "death grip" in which the phase can not produce the work required to proceed. Like the analogous relationship for
the one phase model this relationship includes the slow initial startup and the stable slope toward release of all tasks. The multiple phase model requires the integration of the RTL code with the specifications and therefore includes more of a "tail" to 100% release. This estimate is based on three estimates by two designers and a design manager.

Internal Precedence Relationship(Prototype Testing) = 0.4/0.5/0.6/0.7/0.8/0.9/0.95/1.0/1.0/1.0

This plot stays above the 45 degree line to prevent an internal "death grip" in which the phase can not produce the work required to proceed. Initially this phase can complete 40% of its tasks. This is because they can write the test program as soon as they receive the specifications from the Product Definition phase (an External Precedence Relationship). The testing can then proceed primarily linearly through the prototype until all the parts are tested (70% completed or released). At this point the remainder of the tasks can be completed (100% available). This estimate is based on four estimates by three test engineers and a process engineer.

Internal Precedence Relationship(Reliability/Quality control) = 1.0/1.0/1.0/1.0/1.0/1.0/1.0/1.0/1.0/1.0

This plot stays above the 45 degree line to prevent an internal "death grip" in which the phase can not produce the work required to proceed. Once the Reliability/Quality Control engineer receives the tested prototype and specifications almost all testing could be done simultaneously (if resources were no constraint, as is assumed here). In the Python project this phase occurred within a few weeks and was performed by a single Reliability/Quality Control engineer. This estimate is based on interviews of test engineers concerning their interaction with the reliability/Quality Control engineers (unavailable for interviewing) and on the electronic mail logs of the interactions of the Prototype Testing engineers and the Reliability/Quality Control engineer throughout the Reliability/Quality Control phase.

External Precedence Relationship(Product Definition to Design) = 0.0/0.1/0.25/0.5/0.65/0.80/0.90/0.95/0.97/1.0/1.0
The definition of a few product parts allows the writing of RTL code to begin and progress significantly based on the use of familiar subsystems. Therefore after the receipt of 10-20% of the product definition the design can accelerate (curve is above 45% line) until the final details of the code await the last major definition pieces, causing the flat "tail" on the curve to 100% available at 90% received. This estimate is based on four estimates by a strategic marketing engineer working in Product Definition, a product architect working in Product Definition, a design manager and a design engineer.

External Precedence Relationship(Product Definition to Prototype Testing) =
0.0/0.0/0.0/0.0/0.0/0.0/0.0/0.0/0.0/0.0/0.0/0.0/1.0/1.0

This relationship resembles a sequential interaction except that Prototype Testing is released to do its work when it has received 90% of the product specification. This relationship essentially says that the specification must be almost (but not completely) finished before the Prototype Testing can use it. This estimate is based on interviews with two testing engineers.

External Precedence Relationship(Design to Prototype Testing) =
0.01/0.05/0.2/0.4/0.6/0.80/1.0/1.0/1.0/1.0/1.0/1.0/1.0/1.0

This estimate reflects two available work features: 1) the availability of work to Prototype Test engineers based on the release of RTL code by the Design phase and 2) the static delay for the development steps between the release of RTL code and the beginning of Prototype Testing. The estimate of the static delay will be described first.

The Static Delay between Design and Prototype Testing:
Several relatively quick development activities occur after release of RTL code to layout and before Prototype Testing can begin based on the availability of the products of the RTL code. These activities are layout, tape, mask making, and wafer fabrication. These activities are relatively constant in duration and can be estimated as a static delay between the Design phase and the Prototype Testing phase. It is important to remember that this delay influences only the testing of product by the Prototype Testing phase and not the
testing of specifications. The critical path method was used to model these steps and estimate the size of the static delay. The data for this estimate is from interviews with developers in both the Design and Prototype Testing phases.

![Critical Path Method Estimate of Static Delay between Design and Prototype Test Phases: Activity Chart](image)

**Figure 5-8:** Critical Path Method Estimate of Static Delay between Design and Prototype Test Phases: Activity Chart

The upper left date for each activity is the starting date. The lower right date is the finishing date. The lower left number is the estimated typical duration in weeks.

![Critical Path Method Estimate of Static Delay between Design and Prototype Test Phases: Activity Timeline](image)

**Figure 5-9:** Critical Path Method Estimate of Static Delay between Design and Prototype Test Phases: Activity Timeline
As shown in the Activity Timeline, the static delay is typically about two months in length (the final activity is the Prototype Testing and not part of the delay). This delay was estimated in the External Precedence Relationship by keeping the curve low until a certain percent of the Design has been released. The remainder of the curve reflects the relatively quick release of additional tasks for testing as the curve increases from near 0% available to 100% available in just half the Design release. This estimate is based on five estimates by a design engineer (upstream developer), three test engineers (in-phase developers), and a development process engineer.

External Precedence Relationship (Design to Reliability/Quality Control) = 0.0/0.0/0.0/0.0/0.0/0.0/0.0/1.0/1.0

This relationship resembles a sequential interaction except that Reliability/Quality Control is released to do its work when it has received 90% of the Design products (in the form of the prototypes). This relationship essentially says that the prototypes must be finished before Reliability/Quality Control can use them. This estimate is based on interviews with design and testing engineers.

External Precedence Relationship (Prototype Testing to Reliability/Quality Control) = 0.0/0.0/0.0/0.0/0.0/0.0/0.0/0.0/1.0/1.0

This relationship resembles a sequential interaction except that Reliability/Quality Control is released to do its work when it has received 90% of the Prototype Testing products (in the form of the tested specifications). This relationship essentially says that the specifications must be tested and approved before Reliability/Quality Control can use them. This estimate is based on two estimates by testing engineers.

5.5.3 Comparison of Model Simulations to Python Project Behavior

5.5.3.1 Single Phase Model Behavior

The design activity of the Python project was simulated with the one phase model configuration using the parameter values described previously. Historical behavior of the
Python Design phase was gathered and analyzed to produce reference modes based on the parameters indicated in Figure 5-10.

Figure 5-10: Design Phase Model Calibration to Historical Behavior

Model Parameters with Historical Data Available

Completion dates of the Basework and Rework for the seventeen portions of the RTL code were used to generate reference modes for these rates. The timing of the completion of rework and interviews with developers concerning how rework was managed were used to estimate the Rework required by Internal Quality Assurance rate. Dates when RTL code was released to layout provided data concerning the Release Tasks rate. Finally, by integrating those four inflows and outflows with a spreadsheet the combined
level of the Completed but not Checked and the Hold for Release stocks was calculated over time.

Figures 5-11, 5-12, 5-13 and 5-14 compare the model behavior using the Design phase parameters and the Python Design phase historical behavior.

![Graph showing Design Code Basework Rate over time with Data and Simulation curves, and R² = 94%](image)

**Figure 5-11: Design Phase Model Calibration to Historical Behavior Basework Rate**

The model simulation of Design Phase basework reflects the basic behavioral modes of the historical reference mode. This can be partially attributed to the closeness of the basework reference mode data to the field data collected. Both reflect several significant features:

- the sharp increase in code generation from a relatively slow start
- a maximum basework rate
• the sharp drop in production due to the temporary change in headcount and return to previous production rate when headcount is restored
• the reduction of basework as the majority of the code was completed

The model structure provides possible explanations for the project behavior. Work-availability constraints as described by the Internal Precedence Relationship can restrain early basework. The rate of increasing resources can restrain the rate of increase of basework. The maximum basework rate can be set by the Internal Precedence Relationship. Although the size of such a limit is unclear, discussions with Python designers indicate that such a limit exists because the seventeen code modules could not be worked on simultaneously by seventeen designers, i.e. the process limits progress regardless of available resources. The Internal Precedence Relationship can also describe the decrease in the availability of tasks to work on near the completion of the phase.

Figure 5-12 shows the reference mode and model simulation of the design Phase Rework due to Internal Quality Assurance rate.
Figure 5-12: Design Phase Model Calibration to Historical Behavior
  Rework due to Internal Quality Assurance Rate

The model simulation of Design Phase Rework due to Internal Quality Assurance rate also reflects the basic mode of behavior of the historical reference mode, although not as closely as the Basewrk comparison. The difference can partially be attributed to the number of calculations and assumptions between the field data collected and the reference mode data. Both the simulation and reference mode reflect the rise in error discovery to a peak rate. Both the simulation and historical data also reflect a delay of approximately seven weeks between the completion of a majority of the basewrk and the peak in finding flawed tasks. The earlier increase in the data than the simulation may reflect quality improvement efforts driven by improvement programs instead of the demand for error checking.

Figure 5-13 shows the reference mode and model simulation of the design rework rate.
Figure 5-13: Design Phase Model Calibration to Historical Behavior Rework Rate

The model simulation of the Design Phase Rework rate reflects the behavior mode of the historical reference mode. Both the Rework simulation and reference mode reflect the shapes of the Rework due to Internal Quality Assurance rates with a slight delay of a few weeks. The early increase in the field data relative to the simulation may be due to exogenously driven quality efforts.

Figure 5-12 shows the reference mode and model simulation of the design Tasks Completed but not Checked and Tasks Held for Release.
Figure 5-14: Design Phase Model Calibration to Historical Behavior  
Tasks Completed but not Checked and Tasks Held for Release

The model simulation of Design Phase tasks Completed but not Checked and tasks Held for Release closely reflects the historical behavior. The impacts of the changes in the rates of flow can be seen in the behavior. For example the left side of the plot is dominated by the increase in the Basework rate because few errors have been discovered or corrected. The impact of the exogenous drop in headcount is seen in the flat portion of the curve near week 14. Although error discovery withdraws tasks from the combined stocks the inflow of basework (with some minimal rework) continue to generate increases until approximately week 25. As the number of tasks completed for the first time nears the total scope the discovery and correction of errors begins to dominate the behavior of the stocks. The combined stock decreases for the first time when the error discovery rate exceeds the combined basework and rework rates. The stock increases again as the "wave" of discovered errors is corrected. The second oscillation indicated in the historical
data but not in the simulation is suspected to be due to an informal early release of design work which was found to need rework by a downstream phase and returned. This would not be reflected in a one phase model. After the combined stocks level has remained stable for a period of time the stocks drop to zero when the tasks are released from the holding stock.

5.5.3.2 Causes of Single Phase Model Behavior

Causal loop diagrams can describe the model's structure which generates the model's single phase behavior. Several of the fundamental behaviors are driven by the availability of work. The basic feedback structure is shown in Figure 5-15.

![Diagram](image-url)

**Figure 5-15:** Single Phase Available Work Feedback Structure
The influence of the availability of work in a single phase can be explained through the control and shifting of dominance of the two feedback loops shown in Figure 5-15 above. During most of a phase's duration the positive loop dominates the behavior. This does not necessarily imply that the system grows exponentially during this time. The Internal Precedence Relationship acts as a control, limiting the rate of growth allowed by the positive loop. The stronger the Internal Precedence Relationship Restriction is the weaker the positive loop is. In the case of a very strong restriction such as no additional tasks becoming available even when tasks are being completed and released the positive loop is nullified and dominance is shifted to the negative loop. Eventually the Internal Precedence Relationship reaches 100% and cannot grow further. At this point dominance shifts to the negative feedback loop as tasks are completed and released but no additional tasks become available for basework. The result is a decrease in the Tasks Available for Basework and the Basework rate. The shifting of loop dominance explains the growth in available work and basework at the beginning and middle portions of a phase in response to the dominance of the positive loop and the subsequent decrease in available work and basework once all of the tasks have become available and the dominance shifts to the negative loop.

The rate of increase and decrease in the release of work rates can be explained by the model's structure linking the available work and labor quantity, as shown in figure 5-16.
Figure 5-16: Single Phase Labor Availability Feedback Structure

Figure 5-16 shows how the process description portion of the model influences the resources portions as well as the project progress directly. The rate at which new work becomes available (from the process description) influences the addition of labor by increasing the Labor Required. The delayed change in labor impacts the project progress and thereby the availability of new work. This illustrates the breadth of the influence of the development process due to its driving of the demand for different development activities.

The staggered movement of tasks within the phase is another important feature of the one phase model behavior. Figure 5-17 shows a fundamental structure which helps explain this behavior.
Several types of behavior can be explained with the feedback loops which describe the flow of errors in a single phase. Examples include the lag between basework and quality assurance as shown in Figures 5-11 and 5-12 and the lag between quality assurance and rework as shown in Figures 5-12 and 5-13. The three important positive feedback loops relating these parameters are shown in Figure 5-17. The top positive loop describes the closed loop flow of flawed tasks between the Completed, not Checked stock and the Known Rework stock. The growth or decline in this loop is governed by the entry and exit of tasks from the loop through the Completed, not Checked stock. The lower and right positive feedback loops describe the release of new tasks from the Task List for basework due to the accumulation of Tasks Released and tasks Completed, not Checked.
If only the parameters shown in the loops themselves were active, the stocks and flows would move close to each other in time. However, the six additional parameters represent important delays which cause a "wave" of basework such as would be created by an increase in available tasks for basework to move through the phase's stocks sequentially creating Quality Assurance Demand, generating Quality Assurance work delayed by the Quality Assurance Minimum Duration and Quality Assurance Labor Delay, creating Rework Demand, generating Rework delayed by the Rework Minimum Duration and Rework Labor Delay, and creating more Quality Assurance Demand. One of these "waves" can be seen in both the reference mode data and the model simulation by comparing Figures 5-11, 5-12, and 5-13.

Figure 5-17 also provides an explanation for the large impact of both the Quality Assurance Minimum Duration and Quality Assurance Labor Delay. These two parameters impact two positive feedback loops which influence project progress whereas several other parameters shown in Figure 5-17 influence only one of the three positive loops.

Based on the sensitivity analysis in chapter 3 and the design phase calibration project targets appear to be less influential on model behavior than available work and errors. This can be explained with the compensating feedback loops in the target structures. Figure 5-18 illustrates some of these loops which weaken the influence of schedule targets on performance.
Figure 5-18: Compensating Feedback Loops in Schedule Project Target Structure

The negative feedback loop in Figure 5-18 represents the intended impact, increase workweek to reduce schedule pressure. However the four positive feedback loops resist this policy by reducing actual and expected productivity due to fatigue. These loops weaken the influence of setting an aggressive project cycle time target.

Figure 5-19 illustrates some of the loops which weaken the influence of quality targets on performance.
Figure 5-19: Compensating Feedback Loops in Quality Project Target Structure

The two negative feedback loops in Figure 5-19 represent the intended impact. A gap between the quality goal and condition increase the priority (importance) of applying available labor to looking for errors and correcting errors which are discovered. However, other project features represented by the four positive feedback loops resist the policy by decreasing labor for basework. This slows progress and increases schedule pressure. As described above increased schedule pressure increases errors, causing a decrease in current quality. Compensating feedback loops cause project targets to be relatively ineffective compared to process and labor availability at changing behavior.
5.5.3.3 Multiple Phase Model Behavior

The multiple phase model calibration uses four phases of the Python project: Product Definition, Design, Prototype Testing, and Reliability/Quality Control. Although theoretically every phase can interact with every other phase, this is not reflected in most development processes, including the Python Project. The four phases are linked as shown in Figure 5-2. The progression of the Python project through these four phases is shown in Figure 5-20 below with the model's simulation of the Tasks Released stocks.

![Graph showing multiple phase project simulation](image)

**Figure 5-20: Multiple Phase Project Simulation**

The Design phase begins very soon after the Product Definition phase begins. The difference in the total tasks for the Product Definition phase and the other phases is due to the basis for the number of tasks (specifications versus specifications and RTL code). RTL code can begin before all specifications are completed in concurrent development, causing the simultaneous progress in the Product Definition and Design phases. As previously described the Design phase aggregates and holds completed tasks for release
in relatively large sets. Therefore the stock of design tasks completed and held for release
has also been shown in Figure 5-20. The first and largest release of RTL code from the
Design phase occurs at month 19, causing the Design tasks completed to plummet and the
Design tasks released to jump. This releases Prototype Testing to begin, followed by the
Reliability/Quality Control phase. These five model simulations are compared to
historical behavior in Figures 5-21, 5-22, 5-23, 5-24, and 5-25.

Figure 5-21: Multiple Phase Model Calibration to Historical Behavior
Product Definition Tasks Released
Figure 5-22: Multiple Phase Model Calibration to Historical Behavior
Design Tasks Completed and held for Release

Figure 5-23: Multiple Phase Model Calibration to Historical Behavior
Design Tasks Released
Figure 5-24: Multiple Phase Model Calibration to Historical Behavior
Prototype Testing Tasks Released

Figure 5-25: Multiple Phase Model Calibration to Historical Behavior
Reliability/quality Control Tasks Released
The model simulations reflect the basic behavior patterns of the historical reference modes. Parallel deviations such as in figure 5-25 are partially due to the short durations of some phases. The model structure can help explain the simulated behavior.

5.5.3.4 Causes of Multiple Phase Model Behavior

All of the dynamics of a single phase discussed previously are active in each of the four phases of the multiple phase model. In addition inter-phase dynamics influence the behavior of the multiple phase model. The impacts of policies and processes in earlier phases cascade through subsequent project phases with significant secondary and tertiary impacts. As one of the most influential inter-phase links (see chapter 3) the External Precedence Relationships describe the availability of work in a downstream phase based on the release of tasks by an upstream phase. Figure 5-26 describes how this link can impact the upstream phase as well as the downstream phase.
Figure 5-26: Inter-Phase Available Work Feedback

As is common with negative feedback loops, those in Figure 5-26 can cause the system to tend to oscillate. A small (loose) External Precedence Relationship restriction between Phase 1 and Phase 3 allows many upstream errors to be found by Phase 3. This recycles Phase 1 tasks from the Tasks Released stock to the Known Rework stock for rework. After the delay this causes additional work in Phase 1 and reduces the tasks available for basewark in Phase 3. The result is a reduction in Phase 3 basewark and upstream errors discovered, repeating the oscillation.
However the flow of errors between phases has several other important impacts, as shown in Figure 5-27.

Figure 5-27: Multiple Phase Model Error Flows
Figure 5-27 shows additional impacts of inheriting and returning errors between phases. The release of errors by Phase 1 has several impacts on Phase 1 and Phase 2 beyond altering the availability of work:

- **Increased rework in Phase 1:** Phase 1 errors released and discovered by Phase 2 recycle Phase 1’s released tasks into Phase 1 rework. This increases the labor required to complete Phase 1.

- **Increased error generation by Phase 1:** The increase in Phase 1 rework increases the number of errors generated by Phase 1 since the performance of rework generates errors.

- **Reduced Phase 1 release of Tasks:** Increased rework and error generation in phase 1 slows the rate of task release.

- **Increased rework in Phase 2:** Phase 1 errors released to Phase 2 corrupt Phase 2 tasks. Those corrupted tasks which are discovered by Phase 2 increase Phase 2 rework.

- **Increased error generation by Phase 2:** Released and discovered Phase 1 errors generate Phase 2 errors by creating additional Phase 2 rework. This rework provides additional opportunities to generate errors in Phase 2 which would not have existed if the Phase 1 errors had not created the additional Phase 2 rework.

- **Reduced Phase 2 release of Tasks:** Phase 1 errors released to Phase 2 corrupts Phase 2 tasks. Those corrupted tasks which are recycled to Phase 2 rework are not available for release, thereby reducing the release of Phase 2 tasks.

The impacts shown in Figure 5-27 also have several side effects not shown in the figure:

- **Increased Phase 1 coordination demand:** Released and returned Phase 1 errors generates demand for coordination activity in Phase 1.

- **Reduced labor available for Phase 1 basework and quality assurance:** Increased Phase 1 rework and coordination activities reduce the labor available for basework and quality assurance. This further slows the completion of Phase 1.
• **Increased Phase 2 coordination demand:** Discovering released Phase 1 errors generates demand for coordination activity in Phase 2.

• **Reduced labor available for Phase 2 basework and quality assurance:** Increased Phase 1 rework and coordination activities reduce the labor available for basework and quality assurance. This further slows the completion of Phase 2.

The numerous impacts of error generation also help explain why the 3Basic Probability of Flawed Task is so influential on model behavior.

### 5.5.4 Model Calibration Summary

The Python development project conditions are consistent with the Product Development Project Model boundary assumptions. Parameter values based on field data from the Python development project were used to calibrate the model in single phase and multiple phase configurations. The model simulated the fundamental behavioral characteristics of several project performance data sets reasonably well. Causal loop diagrams were used to explain how model structure causes the model behavior. Generally, available work and resource quantity structures were found to impact behavior more than resource effectiveness and project target structures. The ability of the model to describe the Python project's behavior increases the confidence in the model's use for policy analysis. The next section applies the model to policy analysis within the context of projects like the Python development project.

### 5.6 Policy Analysis

This section demonstrates the use of the Product Development Project Model for analyzing a project management policy. This will be done by investigating the impacts of a single type of management policy, coordination, on project performance. Despite being widely discussed in the business and research literature as a means of improving project performance coordination is not as precisely defined as some policies (e.g. a tax rate) or as well understood. Therefore a brief review of the project coordination literature sets a
context for the analysis. This is followed by a description of how coordination policies can be represented in the Product Development Project Model and the descriptions of three coordination policies. The impacts of those policies on project performance in a project similar to the Python development project will be the basis for a deeper analysis of the effects of specific coordination parameters on performance. The model structure is used to explain the behavior and expand the understanding of coordination. The potential impacts of the analysis for coordination policy design and summaries complete the chapter.

5.6.1 Literature Review of Product Development Project Coordination

5.6.1.1 Traditional Characterizations of Coordination of Product Development Projects

Traditional methods of coordination are based upon a functional sequential product development paradigm for project structure (Watton, 1969; Wheelwright and Clark, 1992; Hayes et al., 1988; Clark and Fujimoto, 1991; Zaccai, 1991; Nevins and Whitney, 1989). The manufacturing development organization associated with the traditional development paradigm is based on separate functional units, typically marketing, engineering or R & D, and manufacturing. Coordination is seen as necessary primarily within the organization. Therefore organizational design and internal coordination mechanisms are the primary methods of coordination within the traditional development paradigm.

While often described as a structure for an entire organization (e.g., Wheelwright and Clark, 1991), the matrix structure can also be applied to individual development projects. Boeing's organization for developing the 727-100 is an example of a matrix structure for a single development project (Maxam, 1978). Technical Staff Engineers represented functional departments. These Technical Staff engineers worked closely with Project Design Engineers. Project Design Engineers represented portions of the aircraft in a manner analogous to project managers representing different development projects. The integration of the functional and project aspects of the aircraft through these two formal
leaders was facilitated by engineering specialists "who each seemed to be a favorable blend of Project Design and Technology Staff experience."

When project managers act as coordinators in the traditional development paradigm, such as in a matrix structure, their role and effectiveness are limited. The role is limited by the perception of coordination as an activity to be performed primarily within the organization. The project manager's effectiveness is limited by their relatively low level of authority when compared to functional department heads. The limitations which low levels of authority place on project managers as coordinators and their effects are illustrated by Clark and Fujimoto (1991b) and Womack et al. (1990). These researchers describe project managers in the traditional paradigm as part of the American and European automobile industry in the 1980s. In one case Womack et al. (1990) describe the contributions of this approach to project management and poor coordination to the two year (40%) delay in the development of General Motors's GM-10 model.

5.6.1.2 Recent Characterizations of Coordination of Product Development Projects

The more recent concurrent cross-functional development paradigm responds in at least two ways to the increased coordination needs caused by increased dynamic environments and interdependencies: expand the scope of coordination and integrate with cross-functional development teams.

The need for increased coordination between development organizations and their environments has been articulated by several researchers. Many participants which are not a part of the immediate project team or product development organization have been identified as requiring coordination, including customers (Bacon, 1994; Fujimoto et al., 1992; Ulrich and Eppinger, 1994; Wheelwright and Clark, 1992 Clark and Fujimoto, 1991b; Wheelwright and Sasser, 1991; Hauser and Clausing, 1988), technology (Iansiti, 1993b, 1992; Iansiti and Clark, 1993; Hayes et al., 1988), the distribution chain (Wheelwright and Clark, 1992), competitors (Bacon, 1994; Wheelwright and Clark, 1992; Hauser and Clausing, 1988), and regulation and standards (Bacon, 1994; Vogel, 1993).
Cross-functional development teams are groups of development specialists from different functional domains who work together on a single development project. Construction projects have utilized cross-functional development teams for many years. The use of cross-functional teams in manufacturing firms has been closely linked to product development performance (Cooper and Kleinschmidt, 1994). As a tool for coordination the formation of cross-functional development teams is an extension of the move away from functional-based groups to the matrix structures used in the traditional development paradigm. Hayes et al. (1988) describe and Wheelwright and Clark (1992) later refine a more detailed model of this shift with intermediate steps defined by the level of influence of project managers.

Boeing's 777 project provides an example of using cross-functional development teams for coordination (Peterson and Sutcliffe, 1992; Stix, 1991). Boeing modified the matrix structure used to develop its 727-100 for the development of the 777 aircraft. Chief Engineers led functional domains such as propulsion, avionics, structures, electrical, the flight deck, and aerodynamics. They were responsible for functionality, reliability, maintainability, manufacturability, cost, and certification. Chief Project Engineers were positioned orthogonal to the Chief Engineers in the matrix structure. They were each responsible for at least one of the airplane's sixty-five individual systems. Additional Chief Project Engineers integrated these individual systems within the airplane as a whole and integrated the development project with external participants such as customers and certification testing. Boeing formed over 270 cross-functional development teams within this structure. Peterson and Sutcliffe's (1992) description of the teams illustrates the cross-functional nature of their role: "These teams are defined around individual airplane systems, and are working cross-systems integration and vertical development issues (life cycle) simultaneously."

5.6.1.3 Coordination Tools, Expected Influences and Benefits

According to the literature product development project coordination can take several forms, including:
• The use of cross-functional development teams with time-dedicated (to the project) members who have balanced stakes in project success and commitment to project success (Cooper and Kleinschmidt, 1994; Rosenthal, 1992; Nevens et al., 1992; Dean and Susman, 1992; Clark and Fujimoto, 1991)

• Strong leadership or project champions (Cooper and Kleinschmidt, 1994; McCord and Eppinger, 1993; Clark and Fujimoto, 1992, 1991)

• Documentation which more clearly defines the development space for all developers. Examples include network diagrams, Quality Function Deployment development checklists or "Lessons Learned" books (Rosenau and Moran, 1993; Bacon et al., 1994; Nevens et al., 1992; Dean and Susman, 1992; Wheelwright and Sasser, 1992; Suh, 1990; Hauser and Clausing, 1988).

• Tools for sharing cross functional knowledge such as collocation and advanced technologies (Murmann, 1994; McCord and Eppinger, 1993; Morelli and Eppinger, 1993; Rosenthal, 1992; Peterson and Sutcliffe, 1992; Nevens et al., 1992; Wheelwright and Clark, 1992; Clark and Fujimoto, 1991)

• Specific positions, teams, or departments dedicated to project coordination (McCord and Eppinger, 1993; Peterson and Sutcliffe, 1992; Nevens et al., 1992; Dean and Susman, 1992; Clark and Fujimoto, 1992)

• Integrating organization structures such as matrix versus functional structures and integrating processes such as systems engineering (Ward et al., 1995; Bacon et al. 1994; McCord and Eppinger, 1993; Peterson and Sutcliffe, 1992)

The literature indicates that these coordination efforts will result in the following changes in a project:
• Earlier agreement on a sharp product definition (Cooper and Kleinschmidt, 1994; Murmann, 1994; Clark and Fujimoto, 1992)

• Increased capability for effective concurrent development, meaning more parallelism in activities will reduce risks and cycle time without degradation of other performance measures (Murmann, 1994; Clark and Fujimoto, 1991)

• Fewer and faster development iterations (Murmann, 1994; Gomory, 1992; Nevens et al., 1992; Wheelwright and Clark, 1992; Clark and Fujimoto, 1991)

• Improved communication in quality, quantity, and timing leading to shared mental models (Clark and Fujimoto, 1991; Rosenthal, 1992; Wheelwright and Clark, 1992; Wheelwright and Sasser, 1992)

• Increased commitment, trust and joint responsibility in the development team (Wheelwright and Clark, 1992; Clark and Fujimoto, 1991)

According to the literature these product development project coordination changes can improve all three primary measures of project performance:

• Reduced cycle time (Cooper and Kleinschmidt, 1994; Iansiti and Clark, 1993; Rosenthal, 1992; Nevens et al., 1992; Wheelwright and Clark, 1992; Clark and Fujimoto, 1991)

• Improved product quality (Iansiti, 1993; Rosenthal, 1992; Nevens et al., 1992; Wheelwright and Clark, 1992; Clark and Fujimoto, 1991)

• Reduced cost (Iansiti, 1993; Nevens et al., 1992; Wheelwright and Clark, 1992; Clark and Fujimoto, 1991)

The case in the literature for improving project performance by increasing coordination is compelling. However the record of attempts to apply coordination policies for improved performance is decidedly mixed. Wheelwright and Clark (1992) cite a case in which unsuccessful cross-functional teams increased cycle times. Reasons for coordination failures cited by other researchers vary. Dean and Susman (1991) found friction between members of the team from design and manufacturing. Wheelwright and Sasser (1991) cite a lack of planning due to a lack of information. Nevens et al. (1991) identified a lack of cross-functional skills in team members and no one taking responsibility for coordination. Clark and Fujimoto (1991b) found an automobile development team
consisting of only liaison people and no developers. The team failed because it was ignored by those developing the product. Contributing to the lack of understanding is the apparently successful coordination of product development projects by other firms without using several of the most commonly cited coordination tools such as collocated dedicated cross-functional teams and frequent meetings (Ward et al., 1995). Some research points to different impacts of coordination of different measures of project performance. Iansiti (1993) found that increased coordination of product development in the automobile industry was related to improvements in quality, cycle time and productivity but not to quality alone. The experience of the Bose Corporation described in the first chapter is an example in which re-engineering the product development process for improved coordination increased one performance measure (quality) but not another (cycle time). Several researchers have identified the need to understand the causal relationships which link coordination to project performance (Adler et al., 1995; Cusumano and Nobeoka; 1991).

Why do some increased coordination policies produce significantly improved project performance while others do not? What impacts do increased coordination have on the internal workings of a project that impact performance?

### 5.6.2 Model Representation of Coordination Policies

Coordination policies can be investigated with the Product Development Project Model by describing different coordination policies with model parameter values and analyzing the resulting project performance through the model structure.

#### 5.6.2.1 Coordination Parameters

Coordination policies are represented in the Product Development Project Model with four parameters:

- **The Coordination Labor Delay** is the gap between the demand for coordination and the application of labor to coordination. It represents the response time of the development team to the need for coordination. This response time would be expected to decrease due to the use of coordination tools such as cross-functional teams, collocation, integrating processes such as regular coordination meetings, and advanced communication technologies.
• **Coordination Priority** is the importance given to the coordination activity relative to basework, quality assurance, and rework. This parameter would be expected to increase with the use of strong project managers, product champions, special coordinating entities and integrating structures and processes.

• **The Coordination Minimum Task Duration** is the average time required to perform a coordination task by a developer within a specific phase. This parameter includes coordination meeting times and integration team activities. This parameter is expected to increase to account for the additional time invested in coordination.

• **The Time to Adjust Coordination Productivity Expectations** describes how quickly developers report the productivity of their coordination efforts and translate them into expectations about future coordination productivity. This parameter would be expected to decrease with increased awareness and focus on coordination.

The literature indicates that adjusting the Product Development Project Model coordination parameters in the directions described above will produce improved project performance in two (cycle time and cost) or all three performance measures.

5.6.2.2 Three Coordination Policies

Three coordination policies will be used for analysis:

• **No Coordination:** This policy prevents coordination activities, allowing the backlog of tasks needing coordination to build up. Although rare or nonexistent in practice, this policy establishes a baseline for comparison of other policy impacts and analysis.

• **Passive Coordination:** This policy reflects a perspective that coordination is a "necessary evil" in development projects. Coordination is left primarily up to individual developers to perform and manage as they see fit. This policy permits coordination work in response to coordination needs but gives it lower priority than basework, quality assurance, or rework and allows significant delays in coordination aspects of the project. This policy closely resembles the Python project in which coordination consisted primarily of the use of a cross-functional development team.

• **Active Coordination:** This policy reflects a perspective that proactive coordination is essential to project success. It considers coordination to be as important as other project development activities. This policy implements coordination tools and otherwise works to reduce delays in performing
coordination and communicating about coordination. As a result more time is spent coordinating tasks needing coordination.

The values for the coordination parameters used to describe the three coordination policies are shown in Table 5-4. They were applied to all four development phases equally.

<table>
<thead>
<tr>
<th>Coordination Parameter</th>
<th>Passive Coordination</th>
<th>No Coordination</th>
<th>Active Coordination</th>
</tr>
</thead>
<tbody>
<tr>
<td>Labor Delay</td>
<td>NA*</td>
<td>6</td>
<td>2</td>
</tr>
<tr>
<td>Priority</td>
<td>0</td>
<td>1</td>
<td>4</td>
</tr>
<tr>
<td>Minimum Tasks Duration</td>
<td>NA*</td>
<td>1.0</td>
<td>1.25</td>
</tr>
<tr>
<td>Adjust Prod Expect. Time</td>
<td>NA*</td>
<td>1.5</td>
<td>0.5</td>
</tr>
</tbody>
</table>

* - Not Applicable. The No Coordination policy precludes any coordination labor. Therefore the Coordination Labor Delay and Coordination Minimum Task Duration parameters have no impact.

Table 5-4: Model Parameter Values for Three Coordination Policies

5.6.3 Project Performance Under Different Coordination Policies

Figures 5-28, 5-29, and 5-30 show the Task Released phase behavior of a project closely resembling the Python Project using the three coordination policies. The relatively early (week 65) first release of design tasks and no rework of returned flawed tasks is evident in the no coordination policy simulation.
Figure 5-28: Project Simulation

No Coordination Policy Project

Figure 5-29: Project Simulation

Passive Coordination Policy Project
The slower completion of all phases and the later initial release of design tasks shown in Figure 5-29 indicate that the passive coordination policy finds some inherited errors.

![Graph showing development tasks over time]

**Figure 5-30: Project Simulation**  
*Active Coordination Policy Project*

The active coordination policy most clearly shows the impacts of inherited errors and coordination on Tasks Released phase behavior. The discovery of inherited errors from the Product Definition, Design and testing phases by the Reliability/Quality Control phase generate coordination and rework in all phases and delay project completion.

Table 5-5 shows the performance of the project under the three coordination policies.
### Coordinated Policy

<table>
<thead>
<tr>
<th>Performance Measure</th>
<th>No Coordination</th>
<th>Passive Coordination</th>
<th>Active Coordination</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cycle Time (weeks)</td>
<td>131</td>
<td>135</td>
<td>140</td>
</tr>
<tr>
<td>Quality (percent defects released from final phase)</td>
<td>54.4</td>
<td>53.2</td>
<td>46.9</td>
</tr>
<tr>
<td>Cost (dollars X $1,000,000)</td>
<td>1.016</td>
<td>1.076</td>
<td>1.203</td>
</tr>
</tbody>
</table>

Table 5-5: Project Performance Using Three Coordinated Policies

#### 5.6.4 Analysis of Project Behavior in Response to Coordination Policies

The project performance shown in Table 5-5 does not confirm to the expectations of much of the literature concerning the impacts of coordination on project performance. Although the literature is not unanimous it generally expects all three performance measures to improve (i.e. measures in Table 5-5 decrease) as coordination increases. Table 5-5 shows a significant (14%) quality improvement but a degradation of cycle time and cost performance. Four points explain this discrepancy. First many of the conclusions of the existing coordination literature are drawn largely from a higher level of aggregation (e.g. Wheelwright and Clark, 1992 at the industry level). Research conclusions based on more aggregate data may be unable to identify differences in types of performance in individual projects. Research based on the project level of aggregation tend to be more limited in their expectations of coordination policies (e.g. Morelli and Eppinger, 1993). Second, some of the conclusions in the existing literature are based primarily on the opinions of industry practitioners (e.g. Cooper and Kleinschmidt, 1994 questionnaire-based research). Conclusions drawn primarily from the opinions of practitioners may reflect the current predictions or expectations of the business phases more than actual experience. Research based on objective data tend to be mixed (e.g. Iansiti, 1993). Third, these large amounts of aggregated and practitioner-based research suggest that
coordination research is relatively young and holds an overly optimistic perspective of the potential of coordination to improve project performance. Fourth, the project conditions assumed by the majority of the research may vary from those used in the analysis above.

Despite its differences with some of the literature the model performance shown in Table 5-5 makes intuitive sense. Coordination is an activity which does not directly contribute to the completion, evaluation, or correction of development tasks in the same direct way that basework, quality assurance, and rework do. Therefore a project with limited resources and more coordination could be expected to spend less resources on development tasks which release tasks and therefore take longer. If costs are related to cycle time, more coordination would also be expected to increase costs. The improvement in quality with increased coordination also makes sense if coordination is seen as a primary means of sharing knowledge within the development team and thereby influencing the generation and finding of errors.

But a deeper understanding of how coordination policies influence performance is needed to design robust coordination policies for improved project performance. Sensitivity analysis of the four coordination parameters used to describe coordination policies revealed that the Coordination Labor Delay and Coordination Priority had significant influence on performance while the Coordination Minimum Task Duration and Time to Adjust Expected Coordination Productivity had little impact. Project performance using both passive and active coordination policies were simulated across a range of labor delay and priority values to analyze their impacts. Cost performance was found to be reflected in cycle time performance. Therefore cycle time and quality will be the focus of this discussion. Project quality performance as Coordination Labor Delay and Coordination Priority vary is shown in Figures 5-31.

---

5.6 I believe that coordination research is following a path similar to that of quality research in which an almost euphoric response to the discovery of a high leverage tool is followed by a slowly growing awareness of the pitfalls and complexity of the issue. Much of coordination research appears to still be in the early euphoric stage.
Figure 5-31: Project Quality Performance versus Coordination Labor Delay and Coordination Priority

Project quality decreases with increasing Coordination Labor Delay for both the passive and active coordination policies. The active policy produces better quality than the passive coordination policy. Project quality improves with increasing Coordination Priority for both the passive and active coordination policies. The active policy produces better quality performance and a slightly steeper improvement with increasing Coordination Priority. The project quality performance changes due to increasing labor delay and priority agree with initial intuition concerning the impact of project structure on system behavior. Increased labor delay is expected to create times when coordination labor is needed but not yet assigned to coordination tasks. This would allow more errors to be generated and released undiscovered. The result would be more released errors and lower quality as the coordination labor delay increases. Increasing coordination priority also generates expected behavior. More coordination would result in fewer errors being generated and more errors being discovered. Therefore the expected response to increasing Coordination Priority is improved quality.
Project cycle time performance as Coordination Labor Delay and Coordination Priority vary is shown in Figure 5-32.

![Graph showing project cycle time performance](image)

**Figure 5-32: Project Cycle Time Performance versus Coordination Labor Delay and Coordination Priority**

Project cycle time increases with increasing Coordination Priority in both the passive and active coordination policies. Both policies produce significant cycle time changes, although the project cycle time under the passive policy changes more than with the active policy case. Project cycle time behaves slightly differently as the Coordination Labor Delay increases under the passive and active coordination policies. Cycle time decreases as labor delays increase under both policies but the passive coordination policy allows the cycle time to increase again slightly as delays grow very long, whereas the active coordination policy continues a slow cycle time decline.

In contrast to quality performance, cycle time behavior is counterintuitive. Both increased Coordination Priority and increased Coordination Labor Delay are expected to increase cycle time. While increased coordination priority does this, increasing the delay in providing labor for needed coordination decreases project cycle time. Delays in shifting labor to a development activity in each development phase is intuitively expected to
cause a delay in the completion of the entire project. But the model behaves very differently by producing decreasing project cycle time as the Coordination Labor Delay increases. Beyond the initial decrease in project cycle time the model behavior reverses itself with slightly increasing cycle times under the passive coordination policy. The next section uses the model structure to explain both the intuitive and counterintuitive behavior described above.

5.6.5 Causes of Project Behavior in Response to Coordination Policies

The model structure can be used to explain the model's behavior. Under the no coordination policy all available labor is used by basework, quality assurance and rework toward the release of tasks. This accelerates the project, reducing cycle time. But the lack of coordination increases the number of errors generated and decreases the number of errors discovered, reducing project quality. These causal links are shown in Figures 5-33, 5-34 and 5-35, which are described in more detail below. The project behavior under the no coordination policy is constant for all values of Coordination Labor Delay and Coordination Priority because the no coordination policy prevents the use of any coordination labor.

The explanation of the model behavior under the passive and active coordination policies begins with the feedback structure shown in Figure 5-33 which describes the intuitive behavior of the project's quality performance shown in Figure 5-31.
Figure 5-33: Causal Relationships Linking Coordination Labor Delay and Coordination Priority to Project Quality

The negative feedback loop in Figure 5-33 seeks to match Coordination Labor Provided to Coordination Labor Required by increasing and decreasing coordination labor in response to coordination demand. Increasing the Coordination Priority and decreasing the Coordination Labor Delay strengthen this loop, thereby speeding up the supply of coordination labor in response to a change in demand. The faster increase in labor increases the Coordination Status, which decreases the generation of errors and increases the percent of existing errors which are discovered. Both these influences decrease the number of errors released, thereby increasing project quality. Therefore increased Coordination Priority increases quality performance and increased Coordination Labor Delay decreases quality performance, as shown in Figure 5-31.
The causal loop diagram in Figure 5-34 explains the increase in cycle time with increasing Coordination Priority shown in Figure 5-32.

![Diagram](image)

**Figure 5-34: Causal Relationships Linking Coordination Priority to Project Cycle Time.**

The Coordination Priority directly impacts the Pressure for Coordination Labor in the negative loop shown in Figure 5-34. This Pressure works with the analogous parameters for basework, quality assurance, and rework to determine which activities get the available labor. As the Coordination Priority increases more of the available labor is used for coordination and therefore less is available for basework, quality assurance, and rework. Since basework, quality assurance, and rework drive the release of tasks more directly than coordination "starving" those activities of needed labor increases cycle time. Increasing Coordination Priority effectively "steals" some of the limited available labor
from basework, quality assurance, and rework for the coordination activity. Therefore increased Coordination Priority increases cycle time as shown in Figure 5-32.

Finally the Figures 5-33 and 5-34 can be combined and expanded to show the model structure which generates the model's counterintuitive behavior in which increasing Coordination Labor Delay decreases cycle time initially and eventually allows cycle time to increase slightly (Figure 5-32).

![Figure 5-35: Causal Links from Coordination Labor Delay to Cycle Time.](image-url)
The initial counterintuitive behavior (decreasing cycle time with increasing delays) can be explained and understood through the interaction of the two feedback loops shown in the upper left portion of Figure 5-35. The negative loop on the left side acts to shift available labor to the coordination activity in response to coordination need as described previously. When the left loop (the coordination labor allocation loop) is dominant labor is "stolen" from basework, quality assurance, and rework for coordination faster than the total labor pool can be increased. A strong coordination labor allocation loop allows the causal links in the lower left portion of Figure 5-35 to dominate, so when the coordination labor allocation feedback loop dominates system behavior cycle times are relatively high. The negative feedback loop on the right side increases the Total Labor Required (for all four development activities) and thereby the Total Labor Provided (after significant delays). When the right loop (the total labor loop) is dominant the labor pool grows fast enough to meet the project's total labor needs without significant "stealing" of labor by coordination from basework, quality assurance, and rework. A strong total labor loop keeps Total Labor Provided high enough to not restrain basework, quality assurance, and rework from releasing tasks. When the total labor feedback loop dominates system behavior cycle times are relatively low. The total labor loop tends to eventually dominate behavior. This is because the "stealing" of labor only shifts the use of the available labor and does not alter the deficit between the Total Labor Required and the Total Labor Provided. Therefore the total labor loop seeks to fill the total labor need regardless of the allocation of labor among the four development activities and the labor allocation loop only temporarily controls the system behavior.

The system behavior becomes clearer when seen as a shifting of the dominance from the labor allocation loop to the total labor loop. Early in a project the labor allocation loop dominates when labor quantity is inadequate. This slows project progress by stealing labor from basework, quality assurance, and rework. But the total labor loop eventually provides the needed labor, allowing all four development activities to proceed unhindered by labor shortage. This tends to accelerate the project. The faster the dominance shifts from the labor allocation loop to the total labor loop, the faster the project will shift from expanding the project cycle time to reducing it. Short Coordination Labor Delays keep the labor allocation loop strong longer, delay the shift in loop dominance to the total labor...
loop, and therefore tend to produce projects with longer cycle times. As the Coordination Labor Delay increases the labor allocation loop becomes weaker, allowing the total labor loop to dominate earlier and the project cycle time to begin shrinking instead of growing sooner. Therefore increasing Coordination Labor Delays cause cycle times to decrease, as shown for Coordination Labor Delays from 2 to 9 weeks in Figure 5-32.

The slight increase in the cycle time as the Coordination Labor Delay grows under the passive coordination policy can be explained with the error generation and error discovery negative feedback loops shown in the right portion of Figure 5-35. When the Coordination Labor Delay is long adequate quantities of labor provided by the total labor loop do not assure adequate coordination labor. This is because the long delay causes the labor allocation loop to be so weak that even when there is enough labor it is not allocated to the coordination activity. Under these conditions the lack of coordination labor allows the Coordination Status to fall, increasing the number of errors generated and decreasing the errors discovered. The result is an increase in coordination (and therefore the total) work required and a resulting increase in cycle time. Since coordination labor is typically only a small portion of the total labor for the project the increase in the cycle time is minimal. This represents a second shift in loop dominance from the total labor loop to the error generation and error discovery loops. Therefore very long Coordination Labor Delays tend to increase cycle times as shown for Coordination Labor Delays above 11 in Figure 5-35. This second shift in loop dominance does not occur under the active coordination policy because the level of coordination does not drop low enough to allow the error loops to dominate.

The preceding hypothesis that the timing of a shift in loop dominance controls the change from increasing to decreasing cycle time can be tested with the Product Development Project Model. The hypothesis is supported if a faster shift in loop dominance from the coordination labor allocation loop to the total labor loop reduces project cycle time. The timing of the shift can be influenced by altering the strength of the total labor loop. If cycle times decrease as the total labor loop strengthens then the hypothesis is supported.

5.7 For clarity each of these two loops represent several feedback loops linking error generation and error discovery to coordination labor. All are negative loops with similar impacts on system behavior.
If cycle times do not decrease as the total labor loop strengthens then a different explanation for the counterintuitive behavior shown in Figure 5-32 should be sought.

The strength of the total labor loop will be varied by changing the maximum Headcount Adjust time constant parameter in the development phases. Figure 5-36 shows cycle time as the total labor loop is strengthened.

![Graph showing cycle time vs. Headcount Adjust time constant](image)

**Figure 5-36: Coordination Policy Hypothesis Test**

The test using the Product Development Project Model shows that cycle time drops as the Headcount Adjust Time decreases (total labor loop strengthens). This supports the hypothesis by showing that faster shifts to the total labor loop decrease cycle time.

### 5.6.6 Impacts of Coordination Policy Analysis on Coordination Policy Design

The differences between the performance due to using no coordination, a passive coordination policy or an active coordination policy shown in Figures 5-31 and 5-32 indicate that the design of a project coordination policy influences project performance. The analysis indicates that the relationships among coordination policy parameters and
performance are neither linear or monotonically increasing or decreasing. These relationships require a more sophisticated understanding of project dynamics than simple rules-of-thumb such as "decrease delays to accelerate a project" to design effective and robust coordination policies. The shifting of dominance among feedback loops provides a means of understanding the impact of structure on behavior.

One implication of the analysis is that more coordination may improve some performance measures while degrading others. This places the policy designer in the common position of being forced to trade improved performance in one domain (e.g. cycle time) for poorer performance in another (e.g. quality) (Rosenau and Moran, 1993; Rosenthal, 1992). The relative values of performance in the time, quality, and cost domains are specific to development products, industries, and organizations. For example the Python project valued cycle time performance very highly and development cost performance less. In contrast many building development projects value development cost more than the other two measures. These considerations are essential to determining realistic performance targets prior to designing a coordination policy. Typically some compromise among the three performance measures is a realistic and desirable project target.

However other impacts of coordination policies may generate improvement in all three performance measures. For example increased coordination may reduce the time required to add new labor through early warning of work loads. Productivities may increase significantly due to increased knowledge and development team morale. Figure 5-37 shows a simulation in which these additional beneficial effects of a coordination policy are included. In comparison to the No Coordination policy results in Table 5-5 this coordination policy improves all three performance measures: cycle time (from 131 weeks to 125 weeks), quality (from 54.4% errors released to 43.5% errors released) and cost (from $1,016,000 to $893,000).
A traditional approach to designing a coordination policy based on the structures described might seek to control the strength of the coordination labor allocation loop to reduce the cycle time delays caused by starving basework, quality assurance and rework of labor. This might entail selecting a compromise between low Coordination Priority and long Coordination Labor Delay to minimize the strongest negative impacts of both caused by their extreme values. However a broader perspective can improve coordination policy design and potentially project performance beyond that possible with a traditional approach. Such an expanded approach could use the understanding of shifting loop dominance to identify and manipulate other high leverage points in the coordination system. For example an alternative to weakening the coordination labor allocation loop to accelerating the shift in loop dominance from the coordination labor allocation loop to the total labor loop could be sought. One alternative is to strengthen the total labor loop. This could be done by shortening the delays in the total labor loop. An example of such a delay is the Headcount Adjustment Time parameter which slows the change in total labor in response to total labor needs. A policy of predicting project labor needs before they actually occur and beginning the search for appropriate personnel could significantly influence this parameter. The model structure can be beneficially used to identify and
investigate a variety of coordination policies which utilize project parameters and relationships beyond those which strictly define coordination activities in the most narrow sense. These investigations can also reveal parameters which may limit the effectiveness of a policy. An example of such a limiting parameter is the Maximum Headcount, which could constrain the effect of reducing the Headcount Adjustment Time parameter. Another advantage of basing coordination policies more broadly is the increased robustness of such policies. An example can be seen in the ability of the active coordination policy in the analysis which activated more coordination parameters to restrain growth in cycle time as Coordination Priority increased when compared to the passive coordination policy which engaged fewer coordination parameters.

5.6.7 Policy Analysis Summary

The analysis of coordination policies using the Product Development Project Model revealed that the impacts of increased coordination differ among the three project performance measures and from much of the current coordination literature. Three coordination policies were simulated: no coordination, passive coordination, and active coordination. As coordination policies became more active project cycle time and cost performance decreased and quality improved. The Coordination Labor Delay and the Coordination Priority were found to be the most influential of the four parameters used to describe coordination policies. Changes in project performance as Coordination Labor Delay and Coordination Priority increase were simulated and revealed counter intuitive cycle time behavior. The model structure was used to explain both the intuitive and counter intuitive behavior. The results of the analysis indicate that coordination policy designers must consciously distinguish among relative values of performance in different domains to set realistic performance targets. The analysis also points to the use of the model structure and shifting loop dominance to expand the search for effective coordination policy parameters and relationships for effective and robust policies.
5.7 The Python Development Project Summary

The Product Development Project Model was calibrated to the Python project, a semiconductor chip development project consistent with the model assumptions. Data concerning the theoretical and practiced development process and organization were integrated into the model structure and parameter estimations for calibration. This required the addition of a model structure describing the aggregation and holding of completed and checked tasks before their release as a group. Two model configurations were calibrated to the Python project. A single phase configuration was calibrated to the design phase of the Python project and a multiple phase configuration was calibrated to the majority of the Python product development process. The calibrated model simulations reflected the fundamental behavior modes of the reference modes of the Python project. The shifting of loop dominance within the model structure provided explanations for the model behavior and expanded the understanding of how the high leverage points of the system identified in chapter 3 influence system behavior.

The model was used to analyze coordination policy impacts on performance measures. Four parameters were used to describe coordination policies. Three coordination policies were tested. As coordination became more active quality improved but cycle time and cost performance degraded. This is intuitive but not in agreement with much of the coordination research literature. The Coordination Labor Delay and Coordination Priority were shown to have significant influence on performance. As these parameters increase quality responded as expected but cycle time decreased with increasing Coordination Labor Delays. The model structure was used to explain both the intuitive and counterintuitive behavior of the system. The coordination policy analysis can have significant impacts on the design of coordination policies and project management. The design of effective and robust coordination policies requires relative valuation of different project performance measures and the setting of realistic targets. Shifting feedback loop dominance can provide an expanded understanding of the relationships between structure and performance and thereby lead to improved coordination policy design.
Chapter 6

Conclusions

6.1 Recapitulation

Successful product development projects are critical to competitiveness in several industries. Changing competitive forces such as globalization, increased customer sophistication, and accelerating technology are increasing the difficulty and leverage of managing product development projects. In response many firms have replaced their functional sequential development paradigm with a more cross-functional concurrent approach. The new paradigm increases the impacts of the relationships among project components on performance and thereby the influence of dynamic project features such as feedback, delays and nonlinear relationships. Successful management of these projects requires an understanding and use of the dynamics of projects. Existing research has focused on a static view of project management, especially concerning the impacts of process structure, where the Critical Path Method paradigm has dominated for decades. This research investigates the impacts of dynamic project structure on performance with a focus on the influences of the development process.
A dynamic simulation model of a multiple phase project was built using the system dynamics methodology. The model integrates several previously developed and tested project structures and adds a separate structure for the development process. Simulations describe the behavior generated by the interaction of customized development phases and a project management structure. Each phase explicitly models the impacts of development processes, resource capacity, scope, and targets on four development activities: basework, quality assurance, rework, and coordination. Project performance is measured in time, quality and cost. The model structure is based on previous project models and field data from a practicing product development organization. Sensitivity tests indicate that the three performance measures are most sensitive to error generation rates and process description parameters. A signal processing model of a small portion of the project model was built and used to investigate the similarities and differences between the system dynamics methodology and a more traditional modeling approach.

The model was calibrated to a computer chip development project for a single development phase configuration and a four phase configuration which represented the majority of the development process. Quantitative and qualitative data concerning the development organization, process, and project was collected for parameter estimation. Project and phase behavior and performance data were collected and analyzed to generate reference modes. Testing revealed that when the model is appropriately parameterized the resulting simulated behavior closely resembles the actual historical behavior of the project. The similarity in behavior modes between the project behavior and model simulations support the model's ability to simulate development project dynamics.

The model was applied to the investigation of coordination policies for improved project performance. Three coordination policies representing different levels of coordination were represented in the model. Model simulations indicate that increased coordination improves quality but can degrade cycle time and cost performance. The model structure helps explain the causes of this behavior. Analysis of the influences of two descriptors of coordination
policy reveal that cycle time can decrease as the delay between coordination labor need and coordination labor provided increases. The model structure helped identify the timing of a shift in feedback loop dominance from the allocation of limited labor among development activities to the provision of adequate labor for all development activities as the cause of this counterintuitive behavior. The use of loop dominance as an alternative to single-link open-loop rules of thumb for describing and managing projects was discussed.

6.2 Major Findings and Discussion

6.2.1 The Role of Development Process in Project Behavior

The characteristics of development processes significantly impact the dynamic behavior of projects. Processes can directly influence project progress by constraining development work. Development processes can also influence progress indirectly by setting the total and relative demands for the labor used in different development activities. The influence of the development process can dominate the influences of some resource structures. In addition, a development process structure which includes recycling of work in a closed loop flow, available work constraints within and between phases and minimum task durations was able to describe complex process impacts on project behavior. The identification of these structures and their dynamic impacts indicate that project models should include dynamic descriptions of development processes.

Based on this finding I speculate that previous dynamic project models which reflect actual behavior have incorporated development processes into resource structures and parameter estimates. Another (albeit unlikely) possibility is that the development process of the projects modeled had no significant impact on behavior. It is possible to describe process constraints which are so loose and resource, scope, and target constraints which are so strict that the process has little influence. However the significant impact of the process structures in the
Product Development Project Model and the differences between the process influences and those of other project components indicate that development processes should be modeled explicitly and separately from resources, targets, or scope to distinguish process impacts.

6.2.2 The Role of Feedback, Delays, and Nonlinear Relationships in Project Behavior

The research shows that projects have many potentially important feedback loops. Some are closed loop flows of work or errors in which pieces of the project leave a position or condition in the project through the performance of development work and return to the condition for repeat performance of the development work. Many other potentially important feedback loops return information about project conditions for use in decision making. These feedback flows of work, errors, and information are dynamic and critical to describing the causal relationships within a project which drive behavior.

Development processes and other project features do not move instantaneously or without bias. Understanding the size and character of the delays which alter these flows is important in relating project structure to behavior. Changing those delays can be a potentially effective tool for improving project performance.

Several of the important relationships which drive project behavior are nonlinear. In particular the relationships which describe the available work constraints within and between phases can be described in significantly greater depth with nonlinear relationships than with linear approximations. These improved descriptions expand the range of project relationships which can be modeled.

6.2.3 Project Constraints

Projects have many constraints on their behavior which resist extremes in performance. The sensitivity analysis produced no fluctuations in project performance over 200%. This is small compared to the performance of many complex systems such as corporate growth or profits
which can experience much larger variations. This characteristic of the model is at least partially due to the large number of negative feedback loops which can influence project behavior. The more negative feedback loops in a model or system the more likely that one of those loops will be activated or become dominant and redirect project behavior when behavior approaches extremes. Another partial explanation is that the size of the system is limited by the number of tasks in the project. As modeled by the Product Development Project Model a project's scope cannot exceed the sum of the phase task lists. This constraint could be relaxed if (for example) the discovery of errors generated more rework than the correction of the flawed task.

6.2.4 An Important Gap in Project Management

There is a gap between the primary methods currently used to describe, model, communicate and manage projects and the complexity of the structures which drive the behavior of those projects. Most project models do not include the impacts of feedback, delays, and nonlinear relationships in the evolution of projects. Those that include some of these project features do not describe the internal structure of a project in adequate detail to identify significant causal relationships. Current project management practice is based on open-loop, single-link linear causal relationships which can be and often are reduced to lists of rules-of-thumb guidelines. The parameters and causal relationships identified by the Python developers is a specific example. Thomsett (1990) is an example of this perspective in the literature for practitioners. These tools are incapable of capturing the dynamic project behavior described by the Product Development Project Model and illustrated by the Python project. Additional tools are needed to extend the bounded rationality of project researchers and managers beyond the capabilities of currently available tools to include dynamic project features, structures and behavior.

Based on this finding I speculate that the gap between the tools currently used to understand and manage projects and the complexity of projects will keep a majority of the existing knowledge concerning managing project complexity trapped in the minds of experienced
project managers and will restrain the development of new knowledge. The lack of tools for
descrribing that knowledge prevents its testing, improvement, and communication to others.
The gap has become a major cause of project performance problems as competitive forces
and the new development paradigm increase project complexity. This finding also implies
that, barring reduced project complexity, project performance cannot significantly improve
without the development of tools for the description, modeling, and management of complex
systems. One of the challenges of bridging this gap is the need for the tool to be useful for
description, modeling, communication and management, not just one or two of these
functions. The next major finding points toward a possible solution.

6.2.5 System Dynamics as a Tool for Researching Projects

The system dynamics methodology and its adjacent tools such as causal loop diagramming
can describe project complexity. The Product Development Project Model is an example of
such a description. The similarity of its simulation of the behavior of the Python project to
field data support its ability to adequately describe dynamic impacts of complex causal
relationships. When combined with the previous finding this indicates that the system
dynamics methodology is a potential tool for bridging the gap between current project tools
and project complexity.

While this research has shown that system dynamics can fill at least part of the research
portion of the gap between current tools and project complexity it has not shown that it can
currently bridge the entire gap. The methodology has proven itself successful as a tool for
investigation and learning but has been applied more narrowly to communication about and
management of projects. The use of changes in loop dominance in the policy analysis portion
of this research to explain counterintuitive project behavior identifies it as a potential tool for
bridging from system dynamics models to effective and efficient communication about
project complexity. I speculate that using system dynamics to describe project complexity
will increase the demand for explanatory and management practice tools such as changing
loop dominance. I further speculate that improved understanding of loop dominance will facilitate meeting that demand.

6.3 Implications for Project Management

The research supports several previous insights about projects such as potential tradeoffs among performance measures and has revealed new insights about the dynamics of projects. Those new insights include:

- Development process dynamics can play as important a role in project behavior as project resources, targets and scope. The dynamic impacts of many project features exceed the ability of traditional project models to describe the impacts of structure on behavior.

- Different development activities within a single development phase influence project behavior differently. The characteristics which distinguished basework, quality assurance, rework and coordination made qualitative differences in project behavior. The distinction between activities which directly impact the completion and release of work and development activities which indirectly influence those activities is particularly important.

- The sizes of delays in dominant project processes have large impacts on project behavior. For example the largest minimum task duration for basework, quality assurance, or rework can constrain progress as development phases increase work rates. Delays in quality assurance processes have particularly large influences due to the many feedback loops which they impact.

- Development processes can trap work within and among development phases with many side effects which cause additional work and degrade performance. High error generation and discovery rates cause many iterations, thereby increasing total work and generally degrading project performance.

- Some project subsystems such as project targets include compensating feedback loops which weaken the effect of attempts to control project behavior. Several of these subsystems describe human reactions to project conditions and management policies.
These insights indicate that development practitioners need to understand and use feedback, delays and nonlinear relationships in the management of development projects. This requires an expansion of the project models used by practitioners. Without ignoring the project features used in traditional project models an expanded project model will focus on different types of project features as shown in Table 6-1.

<table>
<thead>
<tr>
<th>Project Feature</th>
<th>Traditional Project Model</th>
<th>Expanded Project Model</th>
</tr>
</thead>
<tbody>
<tr>
<td>Project Structures described</td>
<td>Static</td>
<td>Static &amp; Dynamic</td>
</tr>
<tr>
<td>Causal Relationships</td>
<td>Open-loop</td>
<td>Feedback</td>
</tr>
<tr>
<td></td>
<td>Single-link</td>
<td>Connected chains of links</td>
</tr>
<tr>
<td>Basis of Development Phase Descriptions</td>
<td>Time (phase duration)</td>
<td>Work (quantity of development tasks)</td>
</tr>
<tr>
<td>Basis of Inter-Phase Relationships</td>
<td>Temporal differences among phase starts and completions</td>
<td>Available work constraints</td>
</tr>
<tr>
<td>Schedule Management focus</td>
<td>Degree of Activity concurrence</td>
<td>Impacts of Changes</td>
</tr>
<tr>
<td>Quality Management focus</td>
<td>Error prevention</td>
<td>Delay sizes and locations</td>
</tr>
<tr>
<td>Resource Management focus</td>
<td>Iteration reduction</td>
<td>Flows of iterative work</td>
</tr>
<tr>
<td></td>
<td>Total resources available to project</td>
<td>Allocation of resources within project</td>
</tr>
</tbody>
</table>

Table 6-1: Management Focus in Traditional and Expanded Project Models

The research also implies that improvements in project performance will be limited if dynamic issues are not addressed. This research supports the existing literature in identifying increased project management difficulty with increased concurrence of development activities. Pressure for improved project performance (particularly reduced development cycle time) are expected to continue the current move toward increased concurrence. The resulting management challenges are primarily dynamic.
6.4 Contributions

6.4.1 A New Research Tool

The Product Development Project Model provides a first attempt to integrate into a testable framework the development process and decision making features of development projects. The result is a new and valuable research tool for investigating the dynamics of projects. The model represents an initial description of how primary project structures interact to dynamically impact project performance. By relating process, resources, scope, and target features to performance the model plays a similar role for dynamic project features as the Critical Path and PERT methods provide for static project features. The model does this by integrating many existing project components into a single project model, introducing and testing several new dynamic project structures, and building a flexible project model. The new structures include:

- **Explicit and separate descriptions of the development process, including:**
  a) The available work precedence relationships describing constraints due to processes both internal and external to each development phase. These relationships allow much deeper constraint descriptions than available in previous models.
  b) The description of the development process with four explicit and separately modeled activities allows improved development phase descriptions.
  c) The minimum task durations allow the unique description of each of the four development activities in each phase.

- **Development process as a generator of demand for development activities:** This is the first primarily demand-driven system dynamics project model. The demand for each of the four development activities directly drives the process limits to progress through the minimum task durations and indirectly drives the resource limits to progress by setting goals for resource structures. In previous models progress is constrained by a resource bottleneck which enlarges and
shrinks due to many factors but does not seek any goal. In this model resources seek to fill a demand for each activity set by the development process. This requires the modeling of the concept that all incomplete work may not be available for development work at any given time and that project developers and managers respond only to the current demand for development and not all future development needs. This differs from previous system dynamics models of projects which assume that all incomplete tasks are available.

- **Closed loop flows of flawed tasks**: These structures are the next step in an evolution of project work model structures which grew out of solely resource based models. This model contributes an explicit stock of work waiting to be corrected with in the closed loop flow of work and the development process descriptions to the work flows.

- **Coflow structures for flawed work**: These structures allow the separate modeling of errors instead of using errors to alter a single flow of work. This allows more explicit and detailed modeling of the causes of error generation and discovery and their impacts.

- **Two-directional flow of error information between phases**: Error information is passed in both the downstream and upstream directions. This impacts both the receiving phases through work corruption and coordination demand and the error generating phase through the return of errors for rework and coordination. This error information passing is used within a flexible dependency network.

- **Generic project structure**: A flexible project model structure allows the modeling of many different types of projects. This is accomplished primarily with two model features. First, a flexible number (up to five) of generic linked development project phases can be customized with parameter values to reflect different development phases. Second, a structure for describing the dependency network among those phases directs the flows of work and errors. Previous system dynamics models of projects have had fixed numbers of phases of fixed relationships among them.

### 6.4.2 Project Dynamics Insights and a Tool for Product Development Practitioners

The insights described in the "Implications for Project Management" section above are also contributions of this research. They illustrate the need for tools which facilitate the expansion of project models by practitioners to include dynamic issues. The Product Development
Project Model is one such tool. The Model can help practitioners improve development project practice by improving the understanding of project dynamics in several ways:

- Small portions of the model can be used to investigate the generic impacts of project structures and changes in project parameters. For example ICI has spent significant time and money to accelerate the checking of development work for errors (reduce their quality assurance minimum task duration). ICI's product development process improvement engineer has used a small version of the model to improve his understanding of the impacts of additional quality assurance task duration reductions in a in a context of multiple process and resource constraints.

- The model can be calibrated and used to improve understandings of the impacts of specific project subsystems on project performance. The investigation of the coordination subsystem in chapter 5 is an example of this type of application.

- The model could be calibrated to specific product development operations and used to design and analyze project management policies. This work could be the basis for the development of improved project management heuristics which include dynamic impacts.

- The model can be revised to focus on a specific type of dynamic behavior and developed into a "management flight simulator" suitable for facilitating learning about the dynamics of projects by a group of product developers and managers.

6.5 Limitations of the Research

The model is designed and built to represent a class of problems (development projects). The variety of projects within that class will always require model calibration to realistically reflect specific projects. The limitations of this model specified in this thesis suggest important issues for the broader application of the model and its underlying concepts within the class of development projects.
• **Model size:** The size and resulting complexity of the model will tend to increase as the model is applied to larger projects. This can be partially addressed by increasing the model's level of aggregation. However this may obscure project features of interest.

• **Level of aggregation:** The level of aggregation of the model and its focus are related. The level of aggregation will tend to increase with project size. Too high a level of aggregation may hide the causal relationships which generate the behavior of interest. In contrast a higher level of aggregation will cause some small variations (e.g. the temporary headcount drop in the Python project) to become irrelevant. Too low a level of aggregation unnecessarily increases the modeling effort and potentially introduces misleading model features. The purpose of the model application and the resulting focus will indicate an appropriate level of aggregation.

• **Data collection for parameterization:** Both larger and smaller projects can raise important data issues. Larger projects will increase the number of phases and sources of data. How to effectively collect and integrate that data from potentially different forms into usable information must be addressed. This challenge may be partially ameliorated by more formal data collection and documentation procedures used in larger projects. Smaller projects tend to generate challenges in collecting data for which no formal or documented trail exists. This requires additional judgment concerning the role of project components and expanded methods of data collection.

• **Organizational and development culture boundaries:** Development projects which span organizational and cultural boundaries can generate issues concerning how the different organizations and cultures interact which are not addressed here. These issues can be very important in development projects (for example see Ward, 1995) and should be accounted for in the application of the model to projects with significantly different or separate organizations and cultures.

• **Environmental change:** Changes in the project environment can also be significant in development projects. Technology development which precedes product development is an example (Iansiti; 1992, 1993a, b, c, d). Development organization support (Roberts; 1964) and competition among projects (Weil et al.; 1973) for resources may also require additional model structure or special attention to model data.
6.6 Future Research

The findings and limitations of this work point to potentially valuable extensions. They include the investigation of:

- Relative sizes and types of influences of process, resources, targets, and scope subsystems on performance for specific groups of development projects based on industry (e.g. construction, automobiles, etc.), project size or number of phases.

- Relative sizes and types of influences of different development activities on project performance.

- Dynamic impacts of project features and policies on important non-performance measures such as project manageability or developer moral.

- Impacts of more detailed modeling of developer experience levels, types of labor and other aggregated project features

- Relax the model boundary assumptions to include multiple projects, market introduction and product performance, technological and organizational evolution, or market competitors.

- Add model structure to internalize currently exogenous inputs to the model such as resource availability, process descriptors and development activity priority.

6.7 Summary Conclusions

This research addressed the important issue of the causes of dynamic behavior in product development projects by building, testing and applying a dynamic simulation model of a multiple phase project. Feedback, delays and nonlinear relationships were found useful in describing the drivers of dynamic behavior. The concept of product development as a set of interactive demand-driven activities was used to build rich descriptions of causal relationships based on previous research and field data. The strong direct and indirect influences of development processes were identified by explicitly separating development
processes from resources, scope, and targets. The use of model structure to explain project behavior was illustrated by applying the model to a specific type of project management policy. This identified changes in loop dominance as a potentially valuable tool in communicating the impacts of complex structure on behavior.

The research identifies a gap between current project models used for management and the complexity of project structures. A failure to bridge this gap is expected to limit project performance improvement. Expanding the knowledge and understanding of project dynamics is a critical part of meeting this need. The development of new or improved tools for communication and management practice is also expected to be essential to translating improved knowledge and understanding into improved project performance.

This research has contributed insights concerning the dynamics of projects, a tested framework for modeling projects based on demand for development activities, a tool for future research and a tool for improving the understanding of product development practitioners. This work has created opportunities for expanding the study of project dynamics in several potentially valuable directions.

This research has pushed project management toward a broader image of projects and its role in project performance. It points to ways of improving performance through improved understanding of project structure and behavior. Future research will expand and refine the understanding and use of dynamics to manage projects. But the foundation for extending project models to include fundamental dynamics exists today.
References


Appendix 3.1

Model Equations

Model Equation Notes

1. Model equations are in a DYNAMO format. Letters before an equation indicate the type of parameter as follows:
   L - level
   N - initial value of a level
   R - rate
   A - auxiliary
   T - table function
   C - constant

2. To conserve space the phase-arrayed equations have generally been left with passing arguments which represent individual phases as "Phase". Assigning a value to a parameter in this form assigns that value to the parameter for all five phases. Expanding a parameter to vary the values for different phases is done by using five copies of the assignment equation, replacing "Phase" in each with the appropriate phase number. Examples of this customization procedure are illustrated at the External and Internal Precedence Relationship parameters.

3. See Appendix 3.2 Model Parameters for the meanings of specific parameters.

* PARAMETER VALUE ASSIGNMENT EQUATIONS
*==============================================
*
* PROCESS
* ___________
A Release_Trigger_Sensativity(Phase)=0.6
A Release_Hold_Avg_Time(Phase)=1.06
A Complexity(Phase)=10
A Ref_Complexity(Phase)=100
A BW_Min_Task_duration(Phase)=2
A RW_Min_Task_Duration(Phase)=1
A QA_Min_Task_Duration(Phase)=1
A Coord_Min_duration(Phase)=1

* SCOPE
* ======
A Task_List(Phase)=1000

* TARGETS
* =========
* Schedule
A Deadline_Switch=1
A Initial_Proj_Deadline=125
A Time_to_Avg_Exp_Compl_Time(Phase)=1
A Resistance_to_Sched_Slip=2
A Max_DL_CHANGE(Phase)=100
A Max_Proj_DL_CHANGE=100

* Quality
A Quality_Goal_Adjust_Time(Phase)=12
A Initial_Quality_Goal(Phase)=0.9
A Project_Quality_Goal=0.9
A Basic_prob_flawed_Task(Phase)=0.5

* Cost
A Percent_hourly_Labor(Phase)=0.00
A Avg_Straight_Pay(Phase)=25
A Cost_Markup(Phase)=2
A Overtime_Premium(Phase)=0.50
A Proj_Budget=500000

* RESOURCES
* =========
* Gross Labor
A Initial_Headcount(Phase)=0.50
A Headcount_Adjustment_Time(Phase)=8
A Max_Headcount(Phase)=1
A Min_Headcount(Phase)=0.001
A HdctJumpSwitch(Phase)=0  *0 = off and 1 = on
A HdctJumpStartTime(Phase)=1
A HdctJumpStopTime(Phase)=30

* Labor Allocation
A alpha(Phase)=100
A Coord_Priority(Phase)=1
A BW_Priority(Phase)=3
A RW_Priority(Phase)=1
A QA_Priority(Phase)=1
A BW_Labor_Delay(Phase)=2
A RW_Labor_Delay(Phase)=1
A QA_Labor_Delay(Phase)=8
A Coord_Labor_Delay(Phase)=4

* Workweek
A Max_Workweek(Phase)=140
A Normal_Workweek(Phase)=40
A Wrkwk_Avg_Time(Phase)=4

* Experience
A Exper_Assim_Time(Phase)=1
A Avg_New_member_Exper(Phase)=6
A Ref_Exper(Phase)=50

* Productivity
A Ref_Coord_Prdctvty(Phase)=1
A Ref_BW_Prdctvty(Phase)=2
A Ref_RW_Prdctvty(Phase)=1
A Ref_QA_Prdctvty(Phase)=1.75
A BW_Prdctvty_Avg_Time(Phase)=1
A RW_Prdctvty_Avg_Time(Phase)=1
A QA_Prdctvty_Avg_Time(Phase)=1
A Coord_Prdctvty_Avg_Time(Phase)=1
A Min_Exp_Coord_Prde(Phase)=0.1
A Min_Exp_QA_Prde(Phase)=0.1
A Min_Exp_BW_Prde(Phase)=0.1
A Min_Exp_RW_Prde(Phase)=0.1
A Adjust_Expect_BW_Prdctvty_Time(Phase)=1
A Change_Expected_QA_Prdctvty_Time(Phase)=1
A Ch_Expect_Coord_Prdctvty_time(Phase)=1
A QA_Prдctvty_Report_Time(Phase)=1
A Report_Coord_Prдctvty_time(Phase)=1
A Report_BW_Prдctvty_Time(Phase)=1
A Wt_to_Current_BW_Prдctvty(Phase)=1.00
A BW_Prдctvty_Influences_Time(Phase)=1
A Avg_Act_BW_Prдctvty_Time(Phase)=1
A Avg_Act_RW_Prдctvty_Time(Phase)=1
A Avg_Act_QA_Prдctvty_Time(Phase)=1
A Avg_Act_Coord_Prдctvty_Time(Phase)=1
A Ref_Qual_of_Practice(Phase)=5

* MODEL OPERATION
  *
  * ==========================
  C LastPhase=5
  C PhaseNo(1)=1
  C PhaseNo(2)=2
  C PhaseNo(3)=3
  C PhaseNo(4)=4
  C PhaseNo(5)=5
  A CloseEnough=0.01
  A Test_Input_1(Phase)=1 * ITERATION SWITCH
  A Test_Input_3(Phase)=1 * NOT USED
  A Test_Input_2(Phase)=0 * RELEASE "DUMP" SWITCH
  A QA_STATUS(Phase)=1 * switches off when Resources included
  A COORD_STATUSTest(Phase)=0.5 * switches off when Resources included
  A ResourceSwitch(Phase)=1
  A Budget_Switch=1
  A Quality_Switch=1

* ARRAY STRUCTURES
  *
  * ================
  * USE A 2D MATRIX for each inter-phase variable
  * (i.e. each inter-phase relationship)
  * DIFFERENTIATE MATRIX "COLUMNS" FROM "ROWS" BY CALLING THEM
  * "UP" (upstream phase) and "DOWN" (downstream phase)

* THE PROJECT NETWORK (PHASE DEPENDENCY) MARIX
  *
  * ================
  *
  * value of 1 means the "down" phase depends on the "up" phase.
* i.e. Up feeds down
* value of 0 means the "down" phase does not depend on the "up" phase
* arrange network (i.e. Number the phases) so that smaller numbered phases feed higher numbered phases
* Delcare Dependency Matrices Variables
  A Dep(up,down)=Dependency(up,down)

* DIAGONAL OF MATRIX = 0 TO PREVENT "DEATH GRIP" SELF-DEPENDENCIES
  A Dependency(1,1)=0
  A Dependency(2,2)=0
  A Dependency(3,3)=0
  A Dependency(4,4)=0
  A Dependency(5,5)=0

* UPPER RIGHT HALF OF MATRIX = 0 TO PREVENT "DEATH GRIP" INTERLOCKING
  A Dependency(2,1)=0
  A Dependency(3,1)=0
  A Dependency(4,1)=0
  A Dependency(5,1)=0
  A Dependency(3,2)=0
  A Dependency(4,2)=0
  A Dependency(5,2)=0
  A Dependency(4,3)=0
  A Dependency(5,3)=0
  A Dependency(5,4)=0

* LOWER LEFT HALF OF MATRIX DESCRIBES THE PROJECT'S PHASE DEPENDENCY NETWORK
  A Dependency(1,2)=1
  A Dependency(1,3)=1
  A Dependency(1,4)=0
  A Dependency(1,5)=0
  A Dependency(2,3)=1
  A Dependency(2,4)=0
  A Dependency(2,5)=0
  A Dependency(3,4)=0
  A Dependency(3,5)=0
  A Dependency(4,5)=0

  A TaskListScale(up,down)=Task_List(up)/Task_List(down)
* AVAILABLE-WORK - INTERNAL PRECEDENCE RELATIONSHIPS

A Fraction_Avail_due_to_Int_gate(Phase)=TABHL(T5(*.Phase),Fract_Compl_and_Rel(Phase),0,1,0,10)

* INTERNAL PRECEDENCE RELATIONSHIPS

T T5(*,1)=0.2/0.40/0.60/0.650/0.70/0.750/0.80/0.95/0.98/1.00 * hyperbolic
T T5(*,2)=0.2/0.40/0.60/0.650/0.70/0.750/0.80/0.95/0.98/1.00 * hyperbolic
T T5(*,3)=0.2/0.40/0.60/0.650/0.70/0.750/0.80/0.95/0.98/1.00 * hyperbolic
T T5(*,4)=0/0/0/0/0/0/0/0/0/0 *Closed Gate
T T5(*,5)=0/0/0/0/0/0/0/0/0/0 *Closed Gate

* LIBRARY OF INTERNAL PRECEDENCE RELATIONSHIP TABLES

* CAN DELETE PHASE BY ALLOWING NO BASEWORK

* T T5(*,1)=0/0/0/0/0/0/0/0/0/0 *Closed Gate

* NO INTERNAL GATE

* T T5(*,1)=1/1/1/1/1/1/1/1/1/1 *Completely Open Gate

* LOCKSTEP INTERNAL GATE

* T T5(*,1)=0.1/0.2/0.3/0.4/0.5/0.6/0.7/0.8/0.9/1.00/1.00 * lockstep

* TIGHT LOCKSTEP

* T T5(*,1)=0.05/0.15/0.25/0.35/0.45/0.55/0.65/0.75/0.85/0.95/1.00 * tight lockstep

* FAST LOCKSTEP

* T T5(*,1)=0.2/0.4/0.6/0.8/1.1/1/1/1/1/1/1.00 * Fast lockstep

* 20%-THEN-FREE INTERNAL GATE TO RELEASE CONTROL TO LABOR SECTORS

* T T5(*,1)=0.2/0.2/1.0/1.0/1.0/1.0/1.0/1.0/1.0/1.0/1.00 * 20% then free

* 20%-THEN-FREE INTERNAL GATE TO RELEASE CONTROL TO LABOR SECTORS

* T T5=0.2/0.2/1.0/1.0/1.0/1.0/1.0/1.0/1.0/1.0/1.00/1.00

* COBRA DESIGN PHASE

* T T5(*,1)=0.2/0.40/0.60/0.650/0.70/0.750/0.80/0.95/0.98/1.00

* AVAILABLE-WORK - EXTERNAL PRECEDENCE RELATIONSHIPS

* SET PRECEDENCE RELATIONSHIPS BETWEEN PHASES
* SET INSIDE AN "IF THEN" TO SET INDEPENDENT UPSTREAM  
* CONCURRENCE TO 1.00 (disables independent phase precedences)

A  
Concurrence(up,down)=FIFZE(1.00,TABHL(T6(*,up,down),Fraction_Released(up),0,1,0.10),Dependency(up,down))

* THESE TABLES NOT USED, THEY FILL THE UPPER RIGHT HALF OF THE MATRIX

<table>
<thead>
<tr>
<th>Table</th>
<th>Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>T6(*,1,1)</td>
<td>0.00/0.00/0.00/0.00/0.00/0.00/0.00/0.00/0.00/1.00</td>
</tr>
<tr>
<td>T6(*,2,1)</td>
<td>0.00/0.00/0.00/0.00/0.00/0.00/0.00/0.00/0.00/1.00</td>
</tr>
<tr>
<td>T6(*,2,2)</td>
<td>0.00/0.00/0.00/0.00/0.00/0.00/0.00/0.00/0.00/1.00</td>
</tr>
<tr>
<td>T6(*,3,1)</td>
<td>0.00/0.00/0.00/0.00/0.00/0.00/0.00/0.00/0.00/1.00</td>
</tr>
<tr>
<td>T6(*,3,2)</td>
<td>0.00/0.00/0.00/0.00/0.00/0.00/0.00/0.00/0.00/1.00</td>
</tr>
<tr>
<td>T6(*,3,3)</td>
<td>0.00/0.00/0.00/0.00/0.00/0.00/0.00/0.00/0.00/1.00</td>
</tr>
<tr>
<td>T6(*,4,1)</td>
<td>0.00/0.00/0.00/0.00/0.00/0.00/0.00/0.00/0.00/1.00</td>
</tr>
<tr>
<td>T6(*,4,2)</td>
<td>0.00/0.00/0.00/0.00/0.00/0.00/0.00/0.00/0.00/1.00</td>
</tr>
<tr>
<td>T6(*,4,3)</td>
<td>0.00/0.00/0.00/0.00/0.00/0.00/0.00/0.00/0.00/1.00</td>
</tr>
<tr>
<td>T6(*,4,4)</td>
<td>0.00/0.00/0.00/0.00/0.00/0.00/0.00/0.00/0.00/1.00</td>
</tr>
<tr>
<td>T6(*,5,1)</td>
<td>0.00/0.00/0.00/0.00/0.00/0.00/0.00/0.00/0.00/1.00</td>
</tr>
<tr>
<td>T6(*,5,2)</td>
<td>0.00/0.00/0.00/0.00/0.00/0.00/0.00/0.00/0.00/1.00</td>
</tr>
<tr>
<td>T6(*,5,3)</td>
<td>0.00/0.00/0.00/0.00/0.00/0.00/0.00/0.00/0.00/1.00</td>
</tr>
<tr>
<td>T6(*,5,4)</td>
<td>0.00/0.00/0.00/0.00/0.00/0.00/0.00/0.00/0.00/1.00</td>
</tr>
<tr>
<td>T6(*,5,5)</td>
<td>0.00/0.00/0.00/0.00/0.00/0.00/0.00/0.00/0.00/1.00</td>
</tr>
</tbody>
</table>

* THESE TABLES DEFINE THE EXTERNAL PRECEDENCE RELATIONSHIPS

<table>
<thead>
<tr>
<th>Table</th>
<th>Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>T6(*,1,2)</td>
<td>0.00/0.00/0.00/0.00/0.00/0.00/0.00/0.00/0.00/1.00 * Almost Sequential</td>
</tr>
<tr>
<td>T6(*,1,3)</td>
<td>0.00/0.00/0.00/0.00/0.00/0.00/0.00/0.00/0.00/1.00 * Almost Sequential</td>
</tr>
<tr>
<td>T6(*,2,3)</td>
<td>0.00/0.00/0.00/0.00/0.00/0.00/0.00/0.00/0.00/1.00 * Almost Sequential</td>
</tr>
<tr>
<td>T6(*,3,4)</td>
<td>0.00/0.00/0.00/0.00/0.00/0.00/0.00/0.00/0.00/1.00 * PPS to RelQual</td>
</tr>
<tr>
<td>T6(*,1,4)</td>
<td>0.00/0.00/0.00/0.00/0.00/0.00/0.00/0.00/0.00/1.00 * Delay to 20, JUMP to 100</td>
</tr>
<tr>
<td>T6(*,1,5)</td>
<td>0.00/0.00/0.00/0.00/0.00/0.00/0.00/0.00/0.00/1.00 * Delay to 20, JUMP to 100</td>
</tr>
<tr>
<td>T6(*,2,4)</td>
<td>0.00/0.00/0.00/0.00/0.00/0.00/0.00/0.00/0.00/1.00 * Delay to 20, JUMP to 100</td>
</tr>
<tr>
<td>T6(*,2,5)</td>
<td>0.00/0.00/0.00/0.00/0.00/0.00/0.00/0.00/0.00/1.00 * Delay to 20, JUMP to 100</td>
</tr>
<tr>
<td>T6(*,3,5)</td>
<td>0.00/0.00/0.00/0.00/0.00/0.00/0.00/0.00/0.00/1.00 * Delay to 20, JUMP to 100</td>
</tr>
<tr>
<td>T6(*,4,5)</td>
<td>0.00/0.00/0.00/0.00/0.00/0.00/0.00/0.00/0.00/1.00 * Delay to 20, JUMP to 100</td>
</tr>
</tbody>
</table>

* LIBRARY OF EXTERNAL PRECEDENCE RELATIONSHIPS

* ..........................................................................................

* NO GATE

* T6=1/1/1/1/1/1.0.0000000000/1.0/1.0/1.0/1.0  * No Gate
PARALLEL
T T6=0.00/1.00/1.0/1/1/1.0/1.0/1.0/1.0 * Parallel

LOCKSTEP
T T6=0.00/.1/2/3/4/.5/.6/.7/.8/.9/1.0 * Lockstep

DELAY TO 20% THEN OPEN TO 100% IMMEDIATELY
T T6=0.00/0.00/0.0/1/1/1.0/1.0/1.0/1.0 * Delay to 20%, JUMP to 100%

DELAY TO 10% THEN RAMP TO 100% AT 60%
T T6=0.00/0.00/0.2/0.4/0.6/0.8/1.0/1.0/1.0/1.0 * Delay to 10, ramp to 100

DELAY TO 60% THEN OPEN FULLY TO 100%
T T6=0.00/0.0/0.0/0.0/0.0/0.0/1.0/1.0/1.0/1.0 * Delay to 10% then JUMP to 100%

DELAY TO 50% THEN OPEN 20% per 10%
T T6=0.00/0.00/0.0/0.0/0.0/0.0/0.2/0.4/0.8/0.9/1.00/1.00 * Delay to 50%, ramp

SEQUENTIAL PHASES
T T6=0.00/0.00/0.0/0.0/0.0/0.0/0.00/0.00/0.00/0.00/1.00 * Almost Sequential

S-CURVE
T T6=0.00/0.05/0.15/0.30/0.45/0.525/0.60/0.65/0.85/0.95/1.00 * S Curve

A
Fraction_Avail_due_to__Ext_gates(Phase)=MIN(FIFZE(1.00,Concurrence(1,Phase),Dependency(1,Phase))
,FIFZE(1.00,Concurrence(2,Phase),Dependency(2,Phase)),FIFZE(1.00,Concurrence(3,Phase),Dependency(3,Phase)),FIFZE(1.00,Concurrence(4,Phase),Dependency(4,Phase)),FIFZE(1.00,Concurrence(5,Phase),Dependency(5,Phase)))

CORE DEVELOPMENT ACTIVITIES
L
Known_Rework(Phase)=Known_Rework(Phase)+dt*(RW_due_to_InPhase_QA(Phase)+RW_due_to_Corrupted_tasks(Phase)+RW_due_to__Dwnstrm_QA(Phase)-Rework(Phase))
N Known_Rework(Phase)=0

R RW_due_to_InPhase_QA(Phase)=QA_inspection_rate(Phase)*prob_Task_Flawed_and_Found(Phase)
R \text{RW}_{\text{due to Corrupted tasks}(\text{Phase})} = (\text{Net Corrupted and Found Tasks}(\text{Phase}) - (\text{QA inspection rate}(\text{Phase}) * ((\text{Net Corrupted and Found Tasks}(\text{Phase}) / (\text{QA inspection rate}(\text{Phase}) + 1e-9)) \times \text{prob Task Flawed and Found}(\text{Phase}))))

L \text{Tasks Completed}(\text{Phase}) = \text{Tasks Completed}(\text{Phase}) + dt \times (\text{Basework}(\text{Phase}) + \text{Rework}(\text{Phase}) - \text{Release Tasks}(\text{Phase}) - \text{RW due to InPhase QA}(\text{Phase}) - \text{RW due to Corrupted tasks}(\text{Phase}) - \text{Tasks to Release Hold}(\text{Phase}))
N \text{Tasks Completed}(\text{Phase}) = 0

R \text{RW}_{\text{due to Downstream QA}(\text{Phase})} = \text{Total Err disc by Dn}(\text{Phase})

A \text{QA inspection rate}(\text{Phase}) = \text{MIN}(\text{QA Proc Limit}(\text{Phase}), \text{QA Labor Limit}(\text{Phase}))
R \text{Rework}(\text{Phase}) = \text{MIN}(\text{RW Process Limit}(\text{Phase}), \text{RW Labor Limit}(\text{Phase}))
R \text{Basework}(\text{Phase}) = \text{MIN}(\text{BW Process Limit}(\text{Phase}), \text{BW Labor Limit}(\text{Phase}))

A \text{Coord Limit}(\text{Phase}) = \text{MIN}(\text{Coord Process Limit}(\text{Phase}), \text{Coord Labor Limit}(\text{Phase}))

R \text{Release Tasks}(\text{Phase}) = \text{FIFZE}(\text{QA inspection rate}(\text{Phase}) - \text{RW due to InPhase QA}(\text{Phase}) - \text{RW due to Corrupted tasks}(\text{Phase}), 0, \text{Test Input _2}(\text{Phase}))

L \text{Tasks Released}(\text{Phase}) = \text{Tasks Released}(\text{Phase}) + dt \times (\text{Release Tasks}(\text{Phase}) + \text{Release Tasks from Hold}(\text{Phase}) - \text{RW due to Downstream QA}(\text{Phase}))
N \text{Tasks Released}(\text{Phase}) = 0

A \text{BW Process Limit}(\text{Phase}) = \text{BW Task Avail Gap}(\text{Phase}) / \text{BW Min Task duration}(\text{Phase})

A \text{RW Process Limit}(\text{Phase}) = \text{Known Rework}(\text{Phase}) / \text{RW Min Task Duration}(\text{Phase})
A \text{QA Process Limit}(\text{Phase}) = \text{Tasks Completed}(\text{Phase}) / \text{QA Min Task Duration}(\text{Phase})
A \text{Coord Process Limit}(\text{Phase}) = \text{Coord Backlog}(\text{Phase}) / \text{Coord Min duration}(\text{Phase})

L \text{Coord Backlog}(\text{Phase}) = \text{Coord Backlog}(\text{Phase}) + dt \times (\text{Current Coord added}(\text{Phase}) - \text{Coord Limit}(\text{Phase}))
N \text{Coord Backlog}(\text{Phase}) = 0

A \text{BW Tasks Remaining}(\text{Phase}) = \text{Task List}(\text{Phase}) - (\text{Known Rework}(\text{Phase}) + \text{Tasks Completed}(\text{Phase}) + \text{Tasks Released}(\text{Phase}) + \text{Hold for Release}(\text{Phase}))

A \text{BW Task Avail Gap}(\text{Phase}) = \text{MAX}(0, \text{Tot Tasks Avail}(\text{Phase}) - \text{Tasks Compl and Rel}(\text{Phase}) - \text{Known Rework}(\text{Phase}))
A \text{Tot Tasks Avail}(\text{Phase}) = \text{Task List}(\text{Phase}) \times \text{MIN}(\text{Fraction Avail due to _Int gate}(\text{Phase}), \text{Fraction Avail due to _Ext gates}(\text{Phase}))
A Fraction Released(Phase) = Tasks Released(Phase) / Task List(Phase)

A
Tasks Compl and Rel(Phase) = Tasks Completed(Phase) + Tasks Released(Phase) + Hold for Release(Phase)
A Fraction Compl and Rel(Phase) = Tasks Compl and Rel(Phase) / Task List(Phase)

* RELEASE TASKS AS A GROUP EQUATIONS

L Hold for Release(Phase) = Hold for Release(Phase) + dt * (Tasks to Release Hold(Phase) -
Release Tasks from Hold(Phase))
N Hold for Release(Phase) = 0

R Tasks to Release Hold(Phase) = Test Input 2(Phase) * (QA inspection rate(Phase) -
RW due to InPhase QA(Phase) - RW due to Corrupted tasks(Phase))

R
Release Tasks from Hold(Phase) = Test Input 2(Phase) * Release Trigger(Phase) * ((Hold for Release(Phase))/dt) + Tasks to Release Hold(Phase))

A Release Trigger Task Gate(Phase) = FIFGE(1,0,TIME,65)

A Release Trigger(Phase) = FIFGE(0,1,MAX(Release Hold Avg Stability(Phase),(-
1)*Release Hold Avg Stability(Phase)),Release Trigger Sensativity(Phase)) * Release Trigger Task Gate(Phase)

A Release Hold Avg Stability(Phase) = Release Hold Avg(Phase) - Hold for Release(Phase)

L Release Hold Avg(Phase) = Release Hold Avg(Phase) + dt * (Change in Release Hold Avg(Phase) -
Empty Release Hold Avg(Phase))
N Release Hold Avg(Phase) = 0

R
Change in Release Hold Avg(Phase) = Test Input 2(Phase) * ((Hold for Release(Phase) + Tasks to Release Hold(Phase) - Release Hold Avg(Phase)) / Release Hold Avg Time(Phase))

R Empty Release Hold Avg(Phase) = FIFZE(0,Release Hold Avg(Phase) / dt,Release Trigger(Phase))

A Tasks Compl and Holding(Phase) = Tasks Completed(Phase) + Hold for Release(Phase)

287
* INTERNAL ERRORS

L
Our_Discd_Errors(Phase)=Our_Discd_Errors(Phase)+dt*(Receive_Our_Errors_fr_Dn(Phase)+Disc_Our_Errors(Phase)-Correct_Our_Errors(Phase))
N Our_Discd_Errors(Phase)=0

R Receive_Our_Errors_fr_Dn(Phase)=Total_Err_disc_by_Dn(Phase)

R Disc_Our_Errors(Phase)=RW_due_to_InPhase_QA(Phase)

R Correct_Our_Errors(Phase)=Rework(Phase)*Our_Discd_Error_density(Phase)

L Our_Errors_Released(Phase)=Our_Errors_Released(Phase)+dt*(Release_Errors(Phase)-Receive_Our_Errors_fr_Dn(Phase)+Release_Errors_from_Hold(Phase))
N Our_Errors_Released(Phase)=0

R Release_Errors(Phase)=(Release_Tasks(Phase)*Clean_Task_error_density(Phase))*(1-Test_Input_2(Phase))

L Our_Undiscd_Errors(Phase)=Our_Undiscd_Errors(Phase)+dt*(Generate_Errors(Phase)-Release_Errors(Phase)-Disc_Our_Errors(Phase)-Errors_lost_in_Corrupted_Tasks(Phase)-Errors_to_Release_Hold(Phase))
N Our_Undiscd_Errors(Phase)=0

L
Hold_Errors_for_Release(Phase)=Hold_Errors_for_Release(Phase)+dt*(Errors_to_Release_Hold(Phase)-Release_Errors_from_Hold(Phase))
N Hold_Errors_for_Release(Phase)=0

R
Errors_to_Release_Hold(Phase)=Test_Input_2(Phase)*Clean_Task_error_density(Phase)*Tasks_to_Release_Hold(Phase)

R
Release_Errors_from_Hold(Phase)=Release_Trigger(Phase)*Test_Input_2(Phase)*(Hold_Errors_for_Release(Phase)/dt)

R Generate_Errors(Phase)=(Basework(Phase)+Rework(Phase))*prob_Task_flawed(Phase)
R
Errors_lost_in_Corrupted_Tasks(Phase)=Compl_Task_error_density(Phase)*RW_due_to_Corrupted_tasks(Phase)

A Compl_Task_error_density(Phase)=Our_Undiscd_Errors(Phase)/(Tasks_Completed(Phase)+1e-9)

A Our_Discd_Error_density(Phase)=Our_Discd_Errors(Phase)/(Known_Rework(Phase)+1e-9)

A prob_Task_flawed(Phase)=1-(1-Basic_prob_flawed_Task(Phase))*(1-prob_err_gen_fr_Task_Complexity(Phase))*(1-prob_of_err_gen_by_QoP(Phase))

A prob_err_gen_fr_Task_Complexity(Phase)=TABHL(T8,FIFZE(0,Complexity(Phase)/Ref_Complexity(Phase),Test/Input_1(Phase)),0.2,0.20)
T T8=0.00/0.00/0.05/0.25/0.40/0.5000000000/0.57/0.69/0.78/0.85/0.90

A prob_of_err_gen_by_QoP(Phase)=TABHL(T3,FIFZE(2,Quality_of_Practice(Phase)/Ref_Qual_of_Practice(Phase),Test/Input_1(Phase)),0.1,0.10)
T T3=0.55/0.45/0.36/0.28/0.21/0.15/0.10/0.06/0.03/0.01/0.00

* FIND INTERNAL ERRORS
A
Prob_finding_our_error_if_exists(Phase)=MIN(1,TABHL(T2,QA_for_Find_Error(Phase),0.5,0.50)*Task_Complex_effect_on_Err_disc(Phase)*QoF_Error_Disc_effect(Phase))
T T2=0.00/0.335/0.535/0.685/0.81/0.88/0.925/0.96/0.985/1.00/1.00

A Task_Complex_effect_on_Err_disc(Phase)=TABHL(T9,Complexity(Phase)/Ref_Complexity(Phase),0.2,0.20)
T T9=1.50/1.20/0.95/0.75/0.60/0.50000000/0.45/0.35/0.25/0.20/0.20

A QoF_Error_Disc_effect(Phase)=TABHL(T4,Quality_of_Practice(Phase)/Ref_Qual_of_Practice(Phase),0.2,0.20)
T T4=0.70/0.80/0.88/0.94/0.98/1.0000000/1.3/1.45/1.52/1.55/1.57

*ERROR DENSITIES
* =============
A Rel_Task__error_density(Phase)=Our_Errors_Released(Phase)/(Tasks_Released(Phase)+1e-9)
A Clean_Task_error_density(Phase)=(Compl_Task_error_density(Phase)*(1-
Prob_finding_our_error_if_exists(Phase)))/(1-
Compl_Task_error_density(Phase))+(Compl_Task_error_density(Phase)*(1-
Prob_finding_our_error_if_exists(Phase)))+1e-9

A
prob_Task_flawed_and_Found(Phase)=Prob_finding_our_error_if_exists(Phase)*Compl_Task_error_densit
y(Phase)

* INHERITED ERRORS
* ===============

A
prob_Task_Corrupted_and_found(up,Phase)=prob_find_Up_error__if_exists(Phase)*Rel_Task__error_den
sity(up)*Dependency(up,Phase)

A
Tasks_Corrupted_and_Found(up,Phase)=QA_inspection_rate(Phase)*prob_Task_Corrupted_and_found(up
,Phase)

A
Corrupted_task_discoveries(Phase)=Tasks_Corrupted_and_Found(1,Phase)+Tasks_Corrupted_and_Found(2,Phase)+Tasks_Corrupted_and_Found(3,Phase)+Tasks_Corrupted_and_Found(4,Phase)+Tasks_Corrupted_and_Found(5,Phase)

A
percent_Multiple_Corruption_discoveries(Phase)=(prob_Task_Corrupted_and_found(1,Phase)*prob_Task
_Corrupted_and_found(2,Phase))+(prob_Task_Corrupted_and_found(1,Phase)*prob_Task_Corrupted_and
_found(3,Phase))+(prob_Task_Corrupted_and_found(1,Phase)*prob_Task_Corrupted_and_found(4,Phase)
)+(prob_Task_Corrupted_and_found(1,Phase)*prob_Task_Corrupted_and_found(5,Phase))+(prob_Task_Corrupted_and_found(2,Phase)*prob_Task_Corrupted_and_found(3,Phase))+(prob_Task_Corrupted_and_found(2,Phase)*prob_Task_Corrupted_and_found(4,Phase))+(prob_Task_Corrupted_and_found(2,Phase)*prob_Task_Corrupted_and_found(5,Phase))+(prob_Task_Corrupted_and_found(3,Phase)*prob_Task_Corrupted_and_found(4,Phase))+(prob_Task_Corrupted_and_found(3,Phase)*prob_Task_Corrupted_and_found(5,Phase))+(prob_Task_Corrupted_and_found(5,Phase))

A
Multiple_Corruption_discoveries(Phase)=QA_inspection_rate(Phase)*percent_Multiple_Corruption_disco
veries(Phase)
A Net_Corrupted_and_Found_Tasks(Phase)=Corrupted_task_discoveries(Phase)-
Multiple_Corruption_discoveries(Phase)

A Up_Flawed Tasks_found(up,Phase)=Tasks_Corrupted_and_Found(up,Phase)*TaskListScale(up,Phase)

A

Total_Err_disc_by_Dn(Phase)=Up_Flawed_Tasks_found(Phase,1)+Up_Flawed_Tasks_found(Phase,2)+Up_Flawed_Tasks_found(Phase,3)+Up_Flawed_Tasks_found(Phase,4)+Up_Flawed_Tasks_found(Phase,5)

A

prob_find_Up_error_if_exists(Phase)=MIN(1,Coord_effect_on_Find_Up_Errors(Phase)*QofP_Error_Dis
c_effect(Phase))

A Coord_effect_on_Find_Up_Errors(Phase)=TABHL(T1,Coord_Status(Phase),0,1,0,10)
T T1=0.00/0.015/0.045/0.115/0.265/0.54/0.715/0.84/0.92/0.975/1.00

* PROJECT SCHEDULE
* ================

L Project_Deadline=Project_Deadline+dt*(Proj_Sched_Slip*Deadline_Switch)
N Project_Deadline=Initial_Proj_Deadline

R Proj_Sched_Slip=FIFGE(MIN(Max_Proj_DL_Change,MAX((-1)*Max_Proj_DL_Change,Expected_Proj_Completion_Time-
Project_Deadline)))+Proj_Sched_Press,Resistance_to_Sched_Slip)

A Proj_Sched_Press=FIFGE(MAX((Exp_Compl_Time(LastPhase)-TIME)/(Project_Deadline-
TIME),(Project_Deadline-TIME)/(Exp_Compl_Time(LastPhase)-TIME)),1,Project_Deadline-
TIME,CloseEnough)

A Expected_Proj_Completion_Time=Exp_Compl_Time(LastPhase)

L DL(Phase)=DL(Phase)+dt*Change_DL(Phase)
N DL(Phase)=(PhaseNo(Phase)*Initial_Proj_Deadline)/LastPhase

R Change_DL(Phase)=FIFGE(MIN(Max_DL_Change(Phase),MAX((-1)*Max_DL_Change(Phase),Exp_Compl_Time(Phase)-
DL(Phase)))+Sched_Pressure(Phase),Resistance_to_Sched_Slip)
A Max_Duration(5)=Path5
A Path5=0

A Max_Duration(4)=MAX(Path4,Path45)
A Path4=0
A Path45=Dependency(4,5)*Expected_Duration(5)

A Max_Duration(3)=MAX(Path3,Path34,Path35,Path345)
A Path3=0
A Path34=Dependency(3,4)*Expected_Duration(4)
A Path35=Dependency(3,5)*Expected_Duration(5)
A Path345=(Dependency(3,4)*Dependency(4,5))*(Expected_Duration(4)+Expected_Duration(5))

A Max_Duration(2)=MAX(Path2,Path23,Path24,Path25,Path234,Path235,Path245,Path2345)
A Path2=0
A Path23=Dependency(2,3)*Expected_Duration(3)
A Path24=Dependency(2,4)*Expected_Duration(4)
A Path25=Dependency(2,5)*Expected_Duration(5)
A Path234=(Dependency(2,3)*Dependency(3,4))*(Expected_Duration(3)+Expected_Duration(4))
A Path235=(Dependency(2,3)*Dependency(3,5))*(Expected_Duration(3)+Expected_Duration(5))
A Path245=(Dependency(2,4)*Dependency(4,5))*(Expected_Duration(4)+Expected_Duration(5))
A
Path2345=(Dependency(2,3)*Dependency(3,4)*Dependency(4,5))*(Expected_Duration(3)+Expected_Duration(4)+Expected_Duration(5))

A
Max_Duration(1)=MAX(Path1,Path12,Path13,Path14,Path15,Path123,Path124,Path125,Path134,Path135,Path145,Path1234,Path1235,Path1245,Path1345,Path12345)
A Path1=0
A Path12=Dependency(1,2)*Expected_Duration(2)
A Path13=Dependency(1,3)*Expected_Duration(3)
A Path14=Dependency(1,4)*Expected_Duration(4)
A Path15=Dependency(1,5)*Expected_Duration(5)
A Path123=(Dependency(1,2)*Dependency(2,3))*(Expected_Duration(2)+Expected_Duration(3))
A Path124=(Dependency(1,2)*Dependency(1,4))*(Expected_Duration(2)+Expected_Duration(4))
A Path125=(Dependency(1,2)*Dependency(2,5))*(Expected_Duration(2)+Expected_Duration(5))
A Path134=(Dependency(1,3)*Dependency(3,4))*(Expected_Duration(3)+Expected_Duration(4))
A Path135=(Dependency(1,3)*Dependency(3,5))*(Expected_Duration(3)+Expected_Duration(5))
A Path145=(Dependency(1,4)*Dependency(4,5))*(Expected_Duration(4)+Expected_Duration(5))
A
Path1234=(Dependency(1,2)*Dependency(2,3)*Dependency(3,4))*(Expected_Duration(2)+Expected_Duration(3))+Expected_Duration(4)

A
Path1235=(Dependency(1,2)*Dependency(2,3)*Dependency(3,5))*(Expected_Duration(2)+Expected_Duration(3))+Expected_Duration(5)

A
Path1245=(Dependency(1,2)*Dependency(2,4)*Dependency(4,5))*(Expected_Duration(2)+Expected_Duration(4))+Expected_Duration(5)

A
Path1345=(Dependency(1,3)*Dependency(3,4)*Dependency(4,5))*(Expected_Duration(3)+Expected_Duration(4))+Expected_Duration(5)

A
Path12345=(Dependency(1,2)*Dependency(2,3)*Dependency(3,4)*Dependency(4,5))*(Expected_Duration(2)+Expected_Duration(3)+Expected_Duration(4)+Expected_Duration(5))

* SCHEDULE EFFECTS
* ===========
L Time_spent_to_Date(Phase)=Time_spent_to_Date(Phase)+dt*(Time__Phase_Active(Phase))
N Time_spent_to_Date(Phase)=0

R Time__Phase_Active(Phase)=FIFZE(0,1,StartFlag(Phase)*Not_Done(Phase))

A Not_Done(Phase)=FIFGE(1,0,Task_List(Phase)-Tasks_Released(Phase),0.01)

A StartFlag(Phase)=FIFGE(1,0,Task_List(Phase)-0.01,BW_Tasks_Remaining(Phase))

A Expected_Duration(Phase)=Time_spent_to_Date(Phase)+Time_Required(Phase)

*-------Expected Start Times for Phase Deadlines before Phase has started--------

L Exp_Compl_Time(Phase)=Exp_Compl_Time(Phase)+dt*Change_Exp_Compl_Time(Phase)
N Exp_Compl_Time(Phase)=Expected_Duration(Phase)+Start_Time(Phase)

R Change_Exp_Compl_Time(Phase)=FIFGE((Expected_Duration(Phase)+Start_Time(Phase))-Exp_Compl_Time(Phase))/Time_to_Avg_Exp_Compl_Time(Phase),DL(Phase),Time_Required(Phase),1e-9)

A Start_Time(Phase)=FIFZE(Exp_Start_Time(Phase),Start_Time1(Phase),StartFlag(Phase))
A Exp_Start_Time(1)=0

A Exp_Start_Time(2)=Dependency(1,2)*Expected_Duration(1)

A Exp_Start_Time(3)=MAX(ExpStart3, ExpStart13, ExpStart23, ExpStart123)
A ExpStart3=0
A ExpStart13=Dependency(1,3)*Expected_Duration(1)
A ExpStart23=Dependency(2,3)*Expected_Duration(2)
A ExpStart123=Dependency(1,2)*Expected_Duration(1)+Dependency(2,3)*Expected_Duration(2)

A
A ExpStart4=0
A ExpStart14=Dependency(1,4)*Expected_Duration(1)
A ExpStart24=Dependency(2,4)*Expected_Duration(2)
A ExpStart34=Dependency(3,4)*Expected_Duration(3)
A ExpStart124=Dependency(1,2)*Expected_Duration(1)+Dependency(2,4)*Expected_Duration(2)
A ExpStart134=Dependency(1,3)*Expected_Duration(3)+Dependency(3,4)*Expected_Duration(3)
A
ExpStart1234=Dependency(1,2)*Expected_Duration(1)+Dependency(2,3)*Expected_Duration(2)+Dependency(3,4)*Expected_Duration(3)

A
A ExpStart5=0
A ExpStart15=Dependency(1,5)*Expected_Duration(1)
A ExpStart25=Dependency(2,5)*Expected_Duration(2)
A ExpStart35=Dependency(3,5)*Expected_Duration(3)
A ExpStart45=Dependency(4,5)*Expected_Duration(4)
A ExpStart125=Dependency(1,2)*Expected_Duration(1)+Dependency(2,5)*Expected_Duration(2)
A ExpStart135=Dependency(1,3)*Expected_Duration(1)+Dependency(3,5)*Expected_Duration(3)
A ExpStart145=Dependency(1,4)*Expected_Duration(1)+Dependency(4,5)*Expected_Duration(4)
A
ExpStart1235=Dependency(1,2)*Expected_Duration(1)+Dependency(2,3)*Expected_Duration(2)+Dependency(3,5)*Expected_Duration(3)
A
ExpStart1245=Dependency(1,2)*Expected_Duration(1)+Dependency(2,4)*Expected_Duration(2)+Dependency(4,5)*Expected_Duration(4)
A
ExpStart1345=Dependency(1,3)*Expected_Duration(1)+Dependency(3,4)*Expected_Duration(3)+Dependency(4,5)*Expected_Duration(4)
A
ExpStart12345=Dependency(1,2)*Expected_Duration(1)+Dependency(2,3)*Expected_Duration(2)+Dependency(3,4)*Expected_Duration(3)+Dependency(4,5)*Expected_Duration(4)

--------------------

L Start_Time1(Phase)=Start_Time1(Phase)+dt*Wait_Time(Phase)
N Start_Time1(Phase)=0

R Wait_Time(Phase)=FIFZE(1.0,StartFlag(Phase))

* For testing: sched press is set = 1.00 for "no effect"

A
Sched_Pressure(Phase)=FIFZE(1.0,MIN(10,FIFGE(MAX(Time_Required(Phase)/Time_to_Deadline(Phase), Time_to_Deadline(Phase)/Time_Required(Phase)),1,Time_Required(Phase),CloseEnough)),ResourceSwitch(Phase)*Deadline_Switch)

A Time_to_Deadline(Phase)=FIFGE(0.01,(DL(Phase)-TIME),0,(DL(Phase)-TIME))

A Sched_Error_Disc_effect(Phase)=TABHL(T10,Sched_Pressure(Phase),0,5,0.50)
T T10=0.7/0.9/1.00/1.20/1.60/2.20/3.00/4.00/5.20/6.60/8.20

A
Time_Required(Phase)=FIFZE(Initial_Proj_Deadline/LastPhase,Required_Personweeks(Phase)/(Headcount(Phase)+1e-9),StartFlag(Phase))

A Sched_BW_Press_effect(Phase)=TABHL(TL7,Sched_Pressure(Phase),0,5,0.50)
T TL7=1.00/1.01/1.04/1.07/1.12/1.17/1.25/1.32/1.44/1.58/1.73

A
Sched_Workweek_effect(Phase)=TABHL(TL10,Time_Required(Phase)/Time_to_Deadline(Phase),0,5,0.5)
T TL10=0.97/0.99/1.000000000000000/1.05/1.15/1.30/1.50/1.80/2.20/2.70/3.30

* REPORTING AND TESTING EQUATIONS
* ==-----------------------------==

L Cycle_Time(Phase)=Cycle_Time(Phase)+DT*(Cycle_Time_Change(Phase))
N Cycle_Time(Phase)=0

R Cycle_Time_Change(Phase)=FIFZE(0,1,(StartFlag(Phase)*NotStoppedFlag(Phase)))

A NotStoppedFlag(Phase)=FIFGE(0,1,Tasks_Released(Phase), (Task_List(Phase)-CloseEnough))

A StoppedFlag(Phase)=FIFGE(1,0,Tasks_Released(Phase), (Task_List(Phase)-CloseEnough))

A Percent_Tasks_in_Rework(Phase)=Known_Rework(Phase)/Task_List(Phase)

A
Sum_of_Tasks(Phase)=Tasks_Completed(Phase)+Tasks_Released(Phase)+Known_Rework(Phase)+Hold_for_Release(Phase)

* **LABOR**
* =======
A Total_Hdct=SUM(Headcount)

L Headcount(Phase)=Headcount(Phase)+dt*(Change_Headcount(Phase))
N Headcount(Phase)=Initial_Headcount(Phase)

* ------------------Headcount Jump Equations--------------

R Change_Headcount(Phase)=(HdctJump(Phase)+HdctJumpBack(Phase))+MIN(Max_Headcount(Phase)-Headcount(Phase), (Required_Headcount(Phase)-Headcount(Phase))/Headcount_Adjustment_Time(Phase))

A HdctJump(Phase)=HdctJumpSwitch(Phase)*FIFZE(0,(Min_Headcount(Phase)-Headcount(Phase)))/dt,HdctJumpStartFlag(Phase)*HdctJumpNotStoppedFlag(Phase)

L HdctJumpStore(Phase)=HdctJumpStore(Phase)+dt*ChangeHdctJumpStore(Phase)
N HdctJumpStore(Phase)=0

A ChangeHdctJumpStore(Phase)=FIFZE(Min_Headcount(Phase)-Headcount(Phase)/dt,0,HdctJumpStartTime(Phase)-TIME)

A HdctJumpStartFlag(Phase)=FIFGE(1,0,TIME,HdctJumpStartTime(Phase))

A HdctJumpNotStoppedFlag(Phase)=FIFGE(0,1,TIME,HdctJumpStopTime(Phase))
A HdcJumpBack(Phase)=HdcJumpSwitch(Phase)*FIFZE(-1*HdcJumpStore(Phase)/dt,0,HdcJumpStopTime(Phase)-TIME)

*--------------------------------------------------------
A Gross_Labor(Phase)=Headcount(Phase)*Workweek(Phase)

A BW_Labor(Phase)=Labor_Fraction_to_RW(Phase)*Gross_Labor(Phase)
A RW_Labor(Phase)=Labor_Fraction_to_RW(Phase)*Gross_Labor(Phase)

A Coord_Labor(Phase)=Labor_Fraction_to_Coord(Phase)*Gross_Labor(Phase)

A QA_Labor(Phase)=Labor_Fraction_to_QA(Phase)*Gross_Labor(Phase)
A Required_Headcount(Phase)=FIFZE(0,(Required_Personweeks(Phase)*Budget_Rqrd_Headct_effect)/(Time_to_Deadline(Phase)+0.1)*NotStoppedFlag(Phase))

A Required_Personweeks(Phase)=Total_Labor_Required(Phase)/Normal_Workweek(Phase)

A Total_Labor_Required(Phase)=QA_Labor_Required(Phase)+Coord_Labor_Required(Phase)+BW_Labor_Required(Phase)+RW_Labor_Required(Phase)

* LABOR ALLOCATION
* -------------
A Coord_Labor_Required(Phase)=Coord_Process_Limit(Phase)/Expect_Coord_Prctvty(Phase)
A BW_Labor_Required(Phase)=BW_Process_Limit(Phase)/Expect_BW_Prctvty(Phase)
A RW_Labor_Required(Phase)=RW_Process_Limit(Phase)/Expect_RW_Prctvty(Phase)
A QA_Labor_Required(Phase)=QA_Process_Limit(Phase)/Expected_QA_Prctvty(Phase)

A Labor_Fraction_to_Coord(Phase)=Press_for_Coord(Phase)/Total_Press_for_Actvities(Phase)
A Labor_Fraction_to_BW(Phase)=Press_for_BW(Phase)/Total_Press_for_Actvities(Phase)
A Labor_Fraction_to_RW(Phase)=Press_for_RW(Phase)/Total_Press_for_Actvities(Phase)
A Labor_Fraction_to_QA(Phase)=Press_for_QA(Phase)/Total_Press_for_Actvities(Phase)

A Press_for_BW(Phase)=EXP(((BW_Labor_Required(Phase)*BW_Priority(Phase)*Sched_BW_Press_effect(Phase)*Cost_effect_on_BW_Import)/alpha(Phase))+0.01)
A Press_for_RW(Phase)=Quality_Goal_Switch*(EXP((RW_Labor_Required(Phase)*RW_Priority(Phase)*Quality Gap_effect_on_QARW_priority)/alpha(Phase))+0.01)
A
Press_for_Coord(Phase)=Quality_Goal_Switch*((EXP((Coord_Labor_Required(Phase)*Coord_Priority(Phase)*Qual_Gap_effect_on_Coord_Import)/alpha(Phase)))+0.01)

A
Press_for_QA(Phase)=Quality_Goal_Switch*(EXP((QA_Labor_Required(Phase)*QA_Priority(Phase)*Qual_Gap_effect_on_QARW_priority)/alpha(Phase)))+0.01)

A
Total_Pressure_for_Acivities(Phase)=Press_for_QA(Phase)+Press_for_Coord(Phase)+Press_for_BW(Phase)+Press_for_RW(Phase)+1e-9

* WORKWEEK
* ============

L Avg_Wrkwk(Phase)=Avg_Wrkwk(Phase)+dt*(Avg_Wrkwk_Change(Phase))
N Avg_Wrkwk(Phase)=Normal_Workweek(Phase)

R Avg_Wrkwk_Change(Phase)=(Workweek(Phase)-Avg_Wrkwk(Phase))/Wrkwk_Avg_Time(Phase)

A
Workweek(Phase)=MIN(Normal_Workweek(Phase)*Sched_Workweek_effect(Phase),Max_Workweek(Phase))

* EXPERIENCE
* ============

L Cumm_Exper(Phase)=Cumm_Exper(Phase)+dt*(Change_Cum_Exper(Phase))
N Cumm_Exper(Phase)=Avg_New_member_Exper(Phase)*Initial_Headcount(Phase)

R
Change_Cum_Exper(Phase)=FIFGE(Net_Exper_Gain(Phase)/Exper_Assim_Time(Phase),0,Cumm_Exper(Phase),0)

A
Avg_memb_Exper(Phase)=FIFGE(Cumm_Exper(Phase)/Headcount(Phase),Cumm_Exper(Phase),Headcount(Phase),1.0)

A
Exper_Lost(Phase)=(-1)*MIN(0,Change_Headcount(Phase))*Avg_memb_Exper(Phase)

A
Exper_index(Phase)=(0.80)**(LOGN(Cumm_Exper(Phase)/Ref_Exper(Phase))/LOGN(2))
A Net_Exper_Gain(Phase) = Basework(Phase) + New_Memb_Exper_Gain(Phase) - Exper_Lost(Phase)

A

New_Memb_Exper_Gain(Phase) = Avg_New_member_Exper(Phase) \times \text{MAX}(0, \text{Change_Headcount}(\text{Phase}))

A \text{Exper_on_Prdcvtv_effect}(\text{Phase}) = T_{\text{ABHL(TL13,Exper_index(Phase),0.5,0.50)}}
T_{\text{TL13}} = 1.33/1.30/1.24/1.18/1.1/1.00/0.9/0.82/0.76/0.72/0.70

* BASEWORK PRODUCTIVITY

* ================
L Actual\_BW\_Prdcvtv(Phase) = Actual\_BW\_Prdcvtv(Phase) + dt \times \text{Change_Actual\_BW\_Prdcvtv(Phase)}
N Actual\_BW\_Prdcvtv(Phase) = Ref\_BW\_Prdcvtv(Phase)

R

\text{Change_Actual\_BW\_Prdcvtv(Phase)} = (\text{Current\_BW\_Prdcvtv(Phase)} / \text{Avg_Act_BW\_Prdcvtv\_Time(Phase)}) \times \text{Exper_on_Prdcvtv\_effect(Phase)} \times \text{Coord\_effect_on_Prdy(Phase)}

A

\text{BW\_Labor\_Limit(Phase)} = (\text{FIFZE(1e10,Actual\_BW\_Prdcvtv(Phase) \times \text{BW\_Labor(Phase)},\text{ResourceSwitch(Phase)})}) / \text{BW\_Labor\_Delay(Phase)}

L \text{Expect_BW\_Prdcvtv(Phase)} = \text{Expect_BW\_Prdcvtv(Phase)} + dt \times (\text{Change_Expect_BW\_Prdcvtv(Phase)})
N \text{Expect_BW\_Prdcvtv(Phase)} = \text{Ref\_BW\_Prdcvtv(Phase)}

R

\text{Change_Expect_BW\_Prdcvtv(Phase)} = \text{MAX}(\text{Change_Avg_BW\_Prdcvtv(Phase)} + \text{BW\_Prdcvtv\_Influenes(Phase)}, \text{Min\_Exp_BW\_Prdy(Phase)} - \text{Expect_BW\_Prdcvtv(Phase)})

A \text{Change_Avg_BW\_Prdcvtv(Phase)} = \text{(Current\_BW\_Prdcvtv(Phase)} - \text{Expect_BW\_Prdcvtv(Phase)}) / \text{BW\_Prdcvtv\_Avg\_Time(Phase)}

A

\text{BW\_Prdcvtv\_Influenes(Phase)} = (\text{Wt\_to_Current_BW\_Prdcvtv(Phase)} \times \text{Current_BW\_Prdcvtv(Phase)}) + ((1 - \text{Wt\_to_Current_BW\_Prdcvtv(Phase)}) \times \text{Historical\_BW\_Prdcvtv\_Belief(Phase)}) - \text{Expect_BW\_Prdcvtv(Phase)}) / \text{BW\_Prdcvtv\_Influenes\_Time(Phase)}

A \text{Current_BW\_Prdcvtv(Phase)} = \text{Basework(Phase)} / (\text{BW\_Labor(Phase)} + 1e-9)

A \text{Historical\_BW\_Prdcvtv\_Belief(Phase)} = \text{Ref\_BW\_Prdcvtv(Phase)}
* REWORK PRODUCTIVITY
* ===============
L Actual_RW_Prdcvtvty(Phase)=Actual_RW_Prdcvtvty(Phase)+dt*Change_Actual_RW_Prdcvtvty(Phase)
N Actual_RW_Prdcvtvty(Phase)=Ref_RW_Prdcvtvty(Phase)

R
Change_Actual_RW_Prdcvtvty(Phase)=(Current_RW_Prdcvtvty(Phase)/Avg_Act_RW_Prdcvtvty_Time(Phase))*Exper_on_Prdcvtvty_effect(Phase)*Coord_effect_on_Prdy(Phase)

A
RW_Labor_Limit(Phase)=(FIFZE(1e50,Actual_RW_Prdcvtvty(Phase)*RW_Labor(Phase),ResourceSwitch(Phase))/RW_Labor_Delay(Phase)

L Expect_RW_Prdcvtvty(Phase)=Expect_RW_Prdcvtvty(Phase)+dt*(Change.Expect_RW_Prdcvtvty(Phase))
N Expect_RW_Prdcvtvty(Phase)=Ref_RW_Prdcvtvty(Phase)

R
Change.Expect_RW_Prdcvtvty(Phase)=MAX(Change_Avg_RW_Prdcvtvty(Phase),Min_Exp_RW_Prdy(Phase)-Expect_RW_Prdcvtvty(Phase))

A Change_Avg_RW_Prdcvtvty(Phase)=(Current_RW_Prdcvtvty(Phase)-
Expect_RW_Prdcvtvty(Phase))/RW_Prdcvtvty_Avg_Time(Phase)

A Current_RW_Prdcvtvty(Phase)=Rework(Phase)/(RW_Labor(Phase)+1e-9)

* QA PRODUCTIVITY
* ===============
L Actual_QA_Prdcvtvty(Phase)=Actual_QA_Prdcvtvty(Phase)+dt*Change_Actual_QA_Prdcvtvty(Phase)
N Actual_QA_Prdcvtvty(Phase)=Ref_QA_Prdcvtvty(Phase)

R
Change_Actual_QA_Prdcvtvty(Phase)=(Current_QA_Prdcvtvty(Phase)/Avg_Act_QA_Prdcvtvty_Time(Phase))
*Exper_on_Prdcvtvty_effect(Phase)*Coord_effect_on_Prdy(Phase)

A
QA_Labor_Limit(Phase)=(FIFZE(1e50,QA_Labor(Phase)*Actual_QA_Prdcvtvty(Phase),ResourceSwitch(Ph
hase)))/QA_Labor_Delay(Phase)
L
Expected_QA_Prdctvty(Phase)=Expected_QA_Prdctvty(Phase)+dt*(Change_Expect_QA_Prdctvty(Phase))
N Expected_QA_Prdctvty(Phase)=Ref_QA_Prdctvty(Phase)

R
Change_Expect_QA_Prdctvty(Phase)=MAX(Change_Avg_QA_Prdctvty(Phase),Min_Exp_QA_Prdsy(Phase))−Expected_QA_Prdctvty(Phase))
A Change_Avg_QA_Prdctvty(Phase)=(Current_QA_Prdctvty(Phase)−Expected_QA_Prdctvty(Phase))/QA_Prdctvty_Avg_Time(Phase)
A Current_QA_Prdctvty(Phase)=QA_inspection_rate(Phase)/(QA_Labor(Phase)+1e-9)

* COORDINATION PRODUCTIVITY
* ================================================================
L
Actual_Coord_Prdctvty(Phase)=Actual_Coord_Prdctvty(Phase)+dt*Change_Actual_Coord_Prdctvty(Phase)
N Actual_Coord_Prdctvty(Phase)=Ref_Coord_Prdctvty(Phase)

R
Change_Actual_Coord_Prdctvty(Phase)=(Current_Coord_Prdctvty(Phase)/Avg_Act_Coord_Prdctvty_Time(Phase))*Exper_on_Prdctvty_effect(Phase)

A
Coord_Labor_Limit(Phase)=(Coord_Labor(Phase)*Actual_Coord_Prdctvty(Phase))/Coord_Labor_Delay(Phase)
A Coord_effect_on_Prdsy(Phase)=TABHL(TL2,Coord_Status(Phase),0,1,0.10)
T TL2=0.195/0.41/0.575/0.725/0.825/0.89/0.945/0.96/0.975/0.985/1.00
A Coord_effect_on_QofP(Phase)=TABHL(TL3,Coord_Status(Phase),0,2,0.20)
T TL3=0.00/0.06/0.18/0.36/0.6/0.9/1.28/1.66/1.84/1.96/2.00
A
Coord_Status(Phase)=FIFZE(COORD_STATUSist(Phase),Coord_Labor(Phase)/(Coord_Labor_Required(Phase)+1e-9),ResourceSwitch(Phase))
L
Expect_Coord_Prdctvty(Phase)=Expect_Coord_Prdctvty(Phase)+dt*(Change_Expect_Coord_Prdctvty(Phase))
N Expect_Coord_Prdctvty(Phase)=Ref_Coord_Prdctvty(Phase)

R
Change_Expect_Coord_Prdctvty(Phase)=MAX(Change_Avg_QA_Prdctvty(Phase),Min_Exp_Coord_Prdy (Phase)-Expect_Coord_Prdctvty(Phase))

A Change_Avg_Coord_Prdctvty(Phase)=(Current_Coord_Prdctvty(Phase)-
Expect_Coord_Prdctvty(Phase))/Coord_Prdctvty_Avg_Time(Phase)

A Current_Coord_Prdctvty(Phase)=Coord_Limit(Phase)/(Coord_Labor(Phase)+1e-9)

A
Current_Coord_added(Phase)=RW_due_to__Dwnstrm_QA(Phase)+RW_due_to__Corrupted_tasks(Phase)

* COST
* ======
L Phase_Cost_to_Date(Phase)=Phase_Cost_to_Date(Phase)+dt*Cumm_Cost(Phase)
N Phase_Cost_to_Date(Phase)=0

R
OT_Cost(Phase)=(Avg_Straight_Pay(Phase)*Overtime_Premium(Phase))*CostMarkup(Phase)*Over_Ti me(Phase)

R Straight_Cost(Phase)=Avg_Straight_Pay(Phase)*Straight_Time(Phase)*CostMarkup(Phase)

A Cumm_Cost(Phase)=OT_Cost(Phase)+Straight_Cost(Phase)

A Over_Time(Phase)=(Gross_Labor(Phase)-Straight_Time(Phase))*Percent_hourly_Labor(Phase)

A Straight_Time(Phase)=Headcount(Phase)*Normal_Workweek(Phase)

* QUALITY
* ========
L Quality_Goal(Phase)=Quality_Goal(Phase)+dt*(Change_Quality_Goal(Phase))
N Quality_Goal(Phase)=Initial_Quality_Goal(Phase)
R Change_Quality.Goal(Phase)=((Current_Known_Quality(Phase)-Quality.Goal(Phase))/Quality.Goal.Adjust_Time(Phase))

A Current_Known_Quality(Phase)=MAX(0,1-Our_Discard_Error_density(Phase))

A Project_Errors.Released=SUM(Our_Errors.Released)

* COST CONTROL
* ==
A Forecasted_Costs=Avg_Cost*(MAX(Project_Deadline-TIME,0))

A Avg_Cost=Project_Cost_to_Date/(TIME+1e-9)

A Project_Cost_to_Date=SUM(Phase_Cost_to_Date)

A Tot_Exp_Costs=Forecasted_Costs+Project_Cost_to_Date

A Budget_Surplus=(Proj_Budget-Tot_Exp_Costs)*Budget_Switch

A Budget_Status=Budget_Surplus/Proj_Budget

A Budget_Rqrd_Headct.effect=TABHL(TC1,Budget_Status*Budget_Switch,-1.00,0,0.10)
T TC1=0.00/0.05/0.10/0.20/0.25/0.35/0.40/0.50/0.70/0.90/1.00

A Cost_effect_on_BW_Import=TABHL(TC2,Budget_Status*Budget_Switch,-0.0,2.0,0.20)
T TC2=1.87/1.58/1.35/1.17/1.06/1.00/0.98/0.95/0.89/0.81/0.65

* QUALITY OF PRACTICE
* ==
A QA_for_Find_Error(Phase)=FIFZE(QA_STATUS(Phase),QA_Labor(Phase)/(QA_Labor_Required(Phase)+1e-9),ResourceSwitch(Phase))

A Quality_of_Practice(Phase)=FIFZE(Ref_Qual_of_Practice(Phase),Ref_Qual_of_Practice(Phase)*Exper_effect_on_QoP(Phase)*Sched_Qual_of_Prac_effect(Phase)*Fatigue_Qual_of_Prac_effect(Phase)*Coord_effect_on_QoP(Phase),ResourceSwitch(Phase))

A Exper_effect_on_QoP(Phase)=TABHL(T7,Exper_index(Phase),0.5,0.50)
T T7=2.50/2.4/2.2/1.9/1.5/1.00/.8/.6/.5/.4/.35

303
A Sched_Quai_of_Prac_effect(Phase)=TABHL(T11,Sched_Pressure(Phase),1,10,0.90)
T T11=1/0.99/0.97/0.94/0.90/0.85/0.79/0.72/0.64/0.55/0.45

A
Fatigue_Qual_of_Prac_effect(Phase)=TABHL(TL12,Avg_Wrkwk(Phase)/Normal_Workweek(Phase),0,5,0.50)
T TL12=1.05/1.05/1.0000000000/0.98/0.94/0.88/0.80/0.70/0.58/0.44/0.44

* QUALITY CONTROL
* ================
A Percent_RW_goal=1.00-Project_Quality_Goal

A
Proj_Tasks_Compl_and_Rel=SUM(Tasks_Completed)+SUM(Tasks_Released)+SUM(Hold_for_Release)

A Current_Project_Rework_Percent=SUM(Known_Rework)/(Proj_Tasks_Compl_and_Rel+1e-9+SUM(Known_Rework))

A Proj_Quality_Gap=Percent_RW_goal-Current_Project_Rework_Percent*Quality_Goal_Switch

A Qual_Gap_effect_on_Coord_Import=TABHL(TQ1,Proj_Quality_Gap*Quality_Goal_Switch,-1.00,0.00,0.10)
T TQ1=2.10/1.90/1.72/1.56/1.42/1.30/1.20/1.12/1.06/1.02/1.00

A Qual_Gap_effect_on_QARW_priority=TABHL(TQ2,Proj_Quality_Gap*Quality_Goal_Switch,-1.00,0.00,0.10)
T TQ2=2.20/2.14/2.07/1.99/1.90/1.80/1.68/1.54/1.38/1.20/1.00
Appendix 3.2

Model Variables

actual_BW_prdctvty(Phase): the tasks initially completed per developer-hour. tasks per developer-hour.

actual_Coord_prdctvty(Phase): the tasks coordinated per developer-hour. tasks per developer-hour.

actual_QA_prdctvty(Phase): the tasks inspected for flaws per developer-hour. tasks per developer-hour.

actual_RW_prdctvty(Phase): the tasks reworked per developer-hour. tasks per developer-hour.

Adj_est.Expect_BW_prdctvty_Time(Phase): the time for team members to adjust their expected productivity to the current actual productivity of basework development activities. weeks.

alpha(Phase): a variable to keep the pressure values in the exponential functions from growing out of range. dimensionless.

Avg_Act_BW_prdctvty_Time(Phase): time to smooth instantaneous basework productivity to prevent unrealistic high frequency fluctuations. weeks.

Avg_Act_Coord_prdctvty_Time(Phase): time to smooth instantaneous coordination productivity to prevent unrealistic high frequency fluctuations. weeks.

Avg_Act_QA_prdctvty_Time(Phase): time to smooth instantaneous quality assurance productivity to prevent unrealistic low frequency fluctuations. weeks.

Avg_Act_RW_prdctvty_Time(Phase): time to smooth instantaneous rework productivity to prevent unrealistic low frequency fluctuations. weeks.

Avg_Cost: the average cost of the project to date for forecasting total project cost. dollars per week.

Avg_memb_Exper(Phase): the average experience level of team members. experience units.
**Appendix 3.2**

**Model Variables**

**Avg_New_member_Exper(Phase):** the amount of useful experience which each new team member brings to the Phase. Experience is measured in units similar to tasks, with the performance of each task generating one more unit of experience. experiences.

**Avg_Straight_Pay(Phase):** the average hourly cost of non-overtime work by a developer without markup for overhead costs. dollars per developer per hour.

**Avg_Wrkwk(Phase):** the average number of hours worked in a week by the team of developers. hours per week.

**Avg_Wrkwk_Change(Phase):** the change in the average workweek. hours per week.

**Basework(Phase):** the initial completion of development tasks in a phase. tasks per week.

**Basic_prob_flawed_Task(Phase):** the probability that a task will be flawed when performed due to the inherent difficulty of the task. dimensionless.

**Budget_Rqrd_Headct_effect:** the reduction in the required (target) headcount to reduce project costs in response to project budget overruns. dimensionless.

**Budget_Status:** the relationship of the project budget and the currently forecasted project costs. dimensionless.

**Budget_Surplus:** the amount which the budget exceeds or is exceeded by the current cost forecast. dollars.

**Budget_Switch:** a model control variable engaging the cost and budget feedback loops.

**BW_Labor(Phase):** the amount of labor applied to basework development activities in a focal phase. person-hours.

**BW_Labor_Delay(Phase):** the time between the need for basewark labor as reflected in the pressure for development activities and when labor allocation shifts in response to that pressure. weeks.

**BW_Labor_Limit(Phase):** the maximum basework rate allowed by the availability and effectiveness of resources for basework. tasks per week.

**BW_Labor_Required(Phase):** the number of full time experienced, rested developers required to complete the currently available basework based on the perceived productivity. developers.

**BW_Min_Task_duration(Phase):** the minimum time required to complete a single task when the task is done the first time, assuming no resource or available-work constraints. weeks.

**BW_Prdctvty_Avg_Time(Phase):** the smoothing time for perceived basework productivity based on the assumption that developers average instantaneous productivity to predict labor needs. weeks.

**BW_Prdctvty_Influences(Phase):** a model control variable used to combine the effects of the historical basework productivity and the expected productivity based on recent work. dimensionless.

**BW_Prdctvty_Influences_Time(Phase):** the smoothing time for expected basework productivity for combination with the effects of historical basework productivity. weeks.
**BW_Priority(Phase):** the importance of applying available labor to basework development activities in relation to the importance of quality assurance or coordination activities. dimensionless.

**BW_Process_Limit(Phase):** the maximum rate of completing tasks the first time they are worked on based upon the process. tasks per hour.

**BW_Tasks_Remaining(Phase):** the number of tasks which have not been worked on for the first time. tasks.

**BW_Task_Avail_Gap(Phase):** the number of tasks which are available to be completed for the first time but have not been completed for the first time. tasks.

**ChangeHdctJumpStore(Phase):** the storage or dumping of the amount which the headcount changed due to an exogenous action. developers.

**Change_Actual_BW_Prdctvty(Phase):** the adjustment of the basework productivity due to the current work. tasks per hour.

**Change_Actual_Coord_Prdctvty(Phase):** the adjustment of the coordination productivity due to the current work. tasks per hour.

**Change_Actual_QA_Prdctvty(Phase):** the adjustment of the quality assurance productivity due to the current work. tasks per hour.

**Change_Actual_RW_Prdctvty(Phase):** the adjustment of the rework productivity due to the current work. tasks per hour.

**Change_Avg_BW_Prdctvty(Phase):** the adjustment of the average basework productivity due to the current actual basework productivity. tasks per hour.

**Change_Avg_Coord_Prdctvty(Phase):** the adjustment of the average coordination productivity due to the current actual coordination productivity. tasks per hour.

**Change_Avg_QA_Prdctvty(Phase):** the adjustment of the average quality control productivity due to the current actual quality control productivity. tasks per hour.

**Change_Avg_RW_Prdctvty(Phase):** the adjustment of the average rework productivity due to the current actual rework productivity. tasks per hour.

**Change_Cum_Exper(Phase):** the net increase or decrease in the total experience of the developers working on a phase of the project. experiences.

**Change_DL(Phase):** the change in the deadline of a phase. weeks.

**Change_Expected_QA_Prdctvty_Time(Phase):** the time for team members to adjust their expected productivity to the current productivity for quality assurance activities. weeks.

**Change_Expect_BW_Prdctvty(Phase):** the increase or decrease in the expected basework productivity based on the current reported basework productivity. tasks per hour per developer.

**Change_Expect_Coord_Prdctvty(Phase):** the increase or decrease in the expected coordination productivity based on the current reported coordination productivity. tasks per hour per developer.
Appendix 3.2

Model Variables

**Change.Expect.QA.Prdctvty(Phase):** the increase or decrease in the expected quality assurance productivity based on the current reported quality assurance productivity. tasks per hour per developer.

**Change.Expect.RW.Prdctvty(Phase):** the increase or decrease in the expected rework productivity based on the current reported rework productivity. tasks per hour per developer.

**Change.Exp.Compl.Time(Phase):** the increase or decrease in the forecasted completion time of a phase. weeks.

**Change.in.Headcount(Phase):** the moving of full time, rested (no fatigue), experienced persons on or off the development team for a single Phase. developers per week.

**Change.in.Release.Hold.Avg(Phase):** the difference between the tasks being held for group release and the average tasks being held for group release. This variable measures the sensitivity of the release trigger. tasks.

**Change.Quality.Goal(Phase):** the movement of the quality goal for the project toward the actual quality experienced in the project. percent defects.

**Ch.Expect.Cood.Prdctvty.time(Phase):** the time for team members to adjust their expected productivity to the current productivity for coordination activities. Set at 4. weeks.

**Clean.Task.error.density(Phase):** the percent of tasks which have been released or are being held for release which contain errors generated by the focal development Phase. This is the number of flawed tasks which were not found by the quality assurance process divided by the sum of those tasks and the unflawed tasks. dimensionless.

**CloseEnough: a model control parameter to prevent extreme and unrealistic behavior when denominators of division portions of model equations approach zero. dimensionless.**

**Complete.BW.Tasks(Phase):** The rate at which tasks are completed the first time they are worked on (basework). tasks per week.

**Complexity(Phase):** the inherent difficulty of performing the tasks in the Phase when compared to other Phases. dimensionless.

**Compl.Task.error.density(Phase):** the percent of tasks which contain flaws and have been completed but not checked for errors. dimensionless.

**Concurrence(up,down):** the percent of the task list of the Phase which could be completed without resource constraints based upon the percent of the Task List which has been released by an upstream Phase (up). dimensionless.

**Coordination.Status(Phase):** the measure of the adequacy of coordination efforts in a Phase, defined by the ratio of the required to actual coordination labor. dimensionless.

**Coord_effect.on.Find.Up.Errors(Phase):** a table function which relates the effect of how well coordination needs are being met on the probability of finding errors inherited from preceding Phases. dimensionless.

**Coord.Labor(Phase):** the amount of labor applied coordination activities in a focal Phase. person-hours.
**Coord_Labor_Delay(Phase):** the time between the need for coordination labor as reflected in the pressure for development activities and when labor allocation shifts in response to that pressure.  weeks.

**Coord_Labor_Limit(Phase):** the maximum coordination rate allowed by the availability and effectiveness of resources for coordination.  tasks per week.

**Coord_Labor_Required(Phase):** the number of full time experienced, rested developers required to complete the currently available coordination based on the perceived productivity.  developers.

**Coord_Limit(Phase):** the rate of coordination work as limited by the development process and coordination labor limits.  tasks per week.

**Coord_Min_duration(Phase):** the minimum time required to coordinate efforts due to finding an inherited error or having a downstream Phase find a flaw inherited from the focal Phase, assuming no resource or available-work constraints.  Set at 13.  weeks.

**Coord_Prdctvty_Avg_Time(Phase):** the smoothing time for perceived coordination productivity based on the assumption that developers average instantaneous productivity to predict labor needs.  weeks.

**Coord_Priority(Phase):** the importance of applying available labor to coordination development activities in relation to the importance of quality assurance or basework development activities.  dimensionless.

**Coord_Process_Limit(Phase):** the maximum coordination rate allowed by the development process.  tasks per week.

**Coord_Status(Phase):** the measure of the adequacy of coordination efforts by comparing the coordination needed and the coordination provided.  dimensionless.

**COORD_STATUS_test(Phase):** a model control parameter which sets the measure of the adequacy of coordination efforts in a Phase.  Switches off when resources are included in model.  dimensionless.

**Coord_effect_on_Prdy(Phase):** a table function which relates the adequacy of coordination efforts to the productivity of work in the Phase.  dimensionless.

**Coord_effect_on_QofP(Phase):** a table function which relates the adequacy of coordination efforts to the Quality of Practice.  dimensionless.

**Correct_Our_Errors(Phase):** the rate at which errors which were generated within the focal development Phase are eliminated by reworking tasks.  tasks per week.

**Corrupted_task_discoveries(Phase):** the number of times that a task was found to be flawed due to an inherited error.  This variable includes repeated findings of the same flawed task when the flaw was induced by more than one error inherited from more than one upstream development task.  tasks.

**Cost_effect_on_BW_Import:** a table function relating the budget status to the importance of performing basework.  Basework priority increases as the budget status worsens.  dimensionless.

**CostMarkup(Phase):** the multiplier of direct labor costs to reflect overhead and other development costs.  dimensionless.
Cumm_Exper_index(Phase): a measure of the growth in the cumulative experience of the development team in the focal Phase, as measured by the ratio of the cumulative experience to the reference experience. This variable is used to find the improvement in productivity due to increased experience. This formulation is based upon the learning curve concept. The learning curve effect is modeled by increasing the current productivity 20% for every doubling of cumulative experience of the product development team in the focal Phase. The model takes current experience, experience lost, and experience gained as input and tracks cumulative experience over the lifetime of the simulation.

Cumm_Cost(Phase): the sum of the straight-time and over-time development costs in a phase. dollars.

Cumm_Exper(Phase): the total accumulated experience of the developers in a phase. experiences.

Current_BW_Prdctvty(Phase): the instantaneous basework productivity in a phase. tasks per week per developer.

Current_Coord_Prdctvty(Phase): the instantaneous coordination productivity in a phase. tasks per week per developer.

Coord_Backlog(Phase): the accumulated demand for coordination work. tasks.

Current_Coord_added(Phase): the current demand for coordination defined as the number of tasks found to be corrupted by inherited errors plus the number of flawed and released tasks which are returned after being discovered by downstream phases. tasks.

Current_Known_Quality(Phase): the percent of total tasks completed initially which are believed to not be flawed. percent.

Current_Project_Rework_Percen: the percent of total tasks completed initially which are known to be flawed. percent.

Current_QA_Prdctvty(Phase): the instantaneous quality assurance productivity in a phase. tasks per week per developer.

Current_RW_Prdctvty(Phase): the instantaneous rework productivity in a phase. tasks per week per developer.

Cycle_Time(Phase): the time between the start of the Phase and the completion of the Phase. weeks.

Cycle_Time_Change(Phase): the indicator that a Phase's cycle time is still accruing, i.e. that the Phase has started but has not yet been completed. weeks.

DeadlineFlag(Phase): an indicator that a the project has passed the deadline of a Phase. dimensionless.

Deadline_Switch: a model control parameter which engages the schedule feedback loops. dimensionless.

Dep(up,down): A dummy variable used to declare the variable Dependency(up,down). dimensionless.
Dependency(up,down): The level of dependency between each pair of Phases. A value of 1 means the "down" Phase depends on the "up" Phase, i.e., Up feeds down. A value of 0 means the "down" Phase does not depend on the "up" Phase. The network is arranged (i.e., numbering of the Phases) so that smaller numbered Phases feed higher numbered Phases. Therefore the diagonal of the matrix = 0 to prevent "death grip" self-dependencies, the upper right half of the matrix = 0 to prevent "death grip" interlocking, and the lower left half of matrix describes dependency network. dimensionless

Disc_Our_Errors(Phase): the rate at which quality assurance efforts find tasks with errors which were generated within the focal development Phase. tasks per week.

DL(Phase): the deadline of a phase as set by the project deadline, the critical path from the phase to the project deadline, and the duration of that critical path. weeks from start.

Empty_Release_Hold_Avg(Phase): the emptying of the averaging stock when the held tasks are released as a group. tasks per week.

Errors_lost_in_Corrupted Tasks(Phase): the rate at which errors generated within the focal development Phase become "lost" because the task which was flawed must be reworked because of an error inherited from an upstream Phase. tasks per week.

Errors_to_Release_Hold(Phase): the flow of tasks with errors which are inspected but are being held with a collection of tasks for release as a group. tasks.

Expected_Duration(Phase): the predicted time between the start and completion of a Phase based upon the number of tasks remaining, the headcount, and the productivity. weeks.

Expected_Proj_Completion_Time: the forecasted completion time is the current time plus the longest duration estimate through all possible critical paths from active phases to the completion of the last phase. weeks from start.

Expected_QA_Prdctvty(Phase): the quality assurance productivity which developers expect to be experienced based on the reported quality assurance productivity. tasks per hour per developer.

Expected_BW_Prdctvty(Phase): the basework productivity which developers expect to be experienced based on the reported basework productivity. tasks per hour per developer.

Expected_Coord_Prdctvty(Phase): the coordination productivity which developers expect to be experienced based on the reported coordination productivity. tasks per hour per developer.

Expected_RW_Prdctvty(Phase): the rework productivity which developers expect to be experienced based on the reported rework productivity. tasks per hour per developer.

Exper_Assim_Time(Phase): the time required to assimilate experience and make it useful to improve productivity. This variable can represent the speed of disseminating new knowledge or learning within the Phase's development team. weeks.

Exper_effect_on_QofP(Phase): a table function describing the impact of the experience of the developers on the quality of their practice. dimensionless.

Exper_index(Phase): the improvement in productivity factor due to increased experience. This formulation is based upon the learning curve concept. The learning curve effect is modeled by increasing
the current productivity 20% for every doubling of cumulative experience of the product development team in the focal Phase. The model takes current experience, experience lost, and experience gained as input and tracks cumulative experience over the lifetime of the simulation.

**Exper_Lost(Phase):** the experience lost to the phase due to the departure of developers. experiences.

**Exper_on_Prdctvty_effect(Phase):** a table function describing the impact of the experience of the developers on their productivity. dimensionless.

**ExpStart123:** the expected start date of phase 1 based on the project deadline and the durations through a critical path. weeks.

**ExpStart1234:** the expected start date of phase 1 based on the project deadline and the durations through a critical path. weeks.

**ExpStart12345:** the expected start date of phase 1 based on the project deadline and the durations through a critical path. weeks.

**ExpStart1235:** the expected start date of phase 1 based on the project deadline and the durations through a critical path. weeks.

**ExpStart124:** the expected start date of phase 1 based on the project deadline and the durations through a critical path. weeks.

**ExpStart1245:** the expected start date of phase 1 based on the project deadline and the durations through a critical path. weeks.

**ExpStart125:** the expected start date of phase 1 based on the project deadline and the durations through a critical path. weeks.

**ExpStart13:** the expected start date of phase 1 based on the project deadline and the durations through a critical path. weeks.

**ExpStart134:** the expected start date of phase 1 based on the project deadline and the durations through a critical path. weeks.

**ExpStart1345:** the expected start date of phase 1 based on the project deadline and the durations through a critical path. weeks.

**ExpStart135:** the expected start date of phase 1 based on the project deadline and the durations through a critical path. weeks.

**ExpStart14:** the expected start date of phase 1 based on the project deadline and the durations through a critical path. weeks.

**ExpStart145:** the expected start date of phase 1 based on the project deadline and the durations through a critical path. weeks.

**ExpStart15:** the expected start date of phase 1 based on the project deadline and the durations through a critical path. weeks.
**ExpStart23**: the expected start date of phase 2 based on the project deadline and the durations through a critical path. weeks.

**ExpStart24**: the expected start date of phase 2 based on the project deadline and the durations through a critical path. weeks.

**ExpStart25**: the expected start date of phase 2 based on the project deadline and the durations through a critical path. weeks.

**ExpStart3**: the expected start date of phase 3 based on the project deadline and the durations through a critical path. weeks.

**ExpStart34**: the expected start date of phase 3 based on the project deadline and the durations through a critical path. weeks.

**ExpStart35**: the expected start date of phase 3 based on the project deadline and the durations through a critical path. weeks.

**ExpStart4**: the expected start date of phase 4 based on the project deadline and the durations through a critical path. weeks.

**ExpStart45**: the expected start date of phase 4 based on the project deadline and the durations through a critical path. weeks.

**ExpStart5**: the expected start date of phase 5 based on the project deadline and the durations through a critical path. weeks.

**Exp_Compl_Time(Phase)**: the time that a Phase is predicted to complete its work based upon the time the current time, the tasks remaining, the headcount, and productivity. weeks from start of project.

**Exp_Start_Time(1)**: the expected start time of phase 1 based on the critical path duration and the project dependency network. weeks.

**Exp_Start_Time(2)**: the expected start time of phase 2 based on the critical path duration and the project dependency network. weeks.

**Exp_Start_Time(3)**: the expected start time of phase 3 based on the critical path duration and the project dependency network. weeks.

**Exp_Start_Time(4)**: the expected start time of phase 4 based on the critical path duration and the project dependency network. weeks.

**Exp_Start_Time(5)**: the expected start time of phase 5 based on the critical path duration and the project dependency network. weeks.

**Fatigue_Qual_of_Prac_effect(Phase)**: a table function which relates the level of Fatigue to the Quality of Practice. 1/hours per week per person.

**Find_Any_Errors_Flag(Phase)**: an indicator if any internally generated errors were found in a Phase. weeks that an error was found.
Find Any Errors Flow(Phase): the rate for the indicator showing if any internally generated errors were found in a Phase. dimensionless.

Flawed and Found Error density(Phase): the percent of tasks which contain flaws and have been found to contain errors. The quality assurance effort is assumed to never consider an unflawed task to be flawed. Therefore this variable is always assumed to be 1.00. dimensionless.

Forecasted Costs: total forecasted costs of all phases. dollars.

Fraction Avail due to Ext gate(up, Phase): the minimum percent of the focal development Phase's task list which could be completed without resource, rework, or speed of processing constraints based upon the percentages of the Task Lists of all upstream development Phases upon which the focal Phase depends and which has been released by those upstream Phases (up).

Fraction Avail due to Int gate(Phase): the percent of the task list which could be completed without resource, rework, or speed of processing constraints based upon the percent of the Task List which is completed and released. dimensionless.

Fraction Released(Phase): the percent of tasks which a development Phase has released to downstream Phases. dimensionless.

Fract Compl and Rel(Phase): the percent of tasks which have been completed and not yet checked for errors or have been completed, checked for errors, and released to downstream development Phases. dimensionless.

Frac Labor in BW(Phase): the percent of basework labor used on basework in the Phase, based upon the relative amount of rework and basework waiting (available) to be done. dimensionless.

Generate Errors(Phase): the rate at which tasks are completed which contain errors. tasks per week.

Gross Labor(Phase): the total labor available to a Phase, based upon the current headcount and workweek. person-hours.

Hdct Jump(Phase): size of instantaneous headcount change due to an exogenous force. developers.

Hdct Jump Back(Phase): size of return to initial headcount prior to instantaneous jump in headcount. developers.

Hdct Jump Not Stopped Flag(Phase): a model control parameter to indicate when the headcount jump is active. dimensionless.

Hdct Jump Start Flag(Phase): a model control parameter to indicate when the headcount jump has started. dimensionless.

Hdct Jump Stop Time(Phase): the time the headcount returns to the initial value. weeks.

Hdct Jump Store(Phase): size of headcount jump. developers.

Hdct Jump Switch(Phase): a model control parameter to engage the headcount jump feature. dimensionless.

Headcount(Phase): the number of full time developers active in the focal development Phase. persons.
Headcount_Adjustment_Time(Phase): the average time required to move a person onto or off of the development team in a focal Phase. weeks.

Historical_BW_Prdctvty_Belief(Phase): the historical belief of developers concerning their productivity in performing basework. tasks per week per developer.

Hold_Errors_for_Release(Phase): the number of flawed tasks which have been completed and checked for errors and are being accumulated for release as a group. tasks.

Hold_for_Release(Phase): the number of tasks which have been completed and checked for errors and are being accumulated for release as a group. tasks.

Initial_Headcout(Phase): headcount at start of Phase. Set at 1.00. persons.

Initial_Proj_Deadline: the project deadline at the beginning of the project. weeks.

Initial_Quality_Goal(Phase): desired percent flawless tasks passed downstream by each Phase. Set at 1.00 (no errors passed). dimensionless.

Known_Rework(Phase): the number of tasks in a development Phase which are known to be flawed and require rework. tasks.

Labor_Fraction_to_BW(Phase): the percent of the labor at any time which is used for basework. dimensionless.

Labor_Fraction_to_Coord(Phase): the percent of the labor at any time which is used for coordination. dimensionless.

Labor_Fraction_to_QA(Phase): the percent of the labor at any time which is used for quality assurance. dimensionless.

Labor_Fraction_to_RW(Phase): the percent of the labor at any time which is used for rework. dimensionless.

LastPhase: identifier of the final Phase in the Phase Dependency Network. dimensionless.

Max_DL_Change(Phase): a model control parameter to prevent extreme and unrealistic schedule pressures as the phase approaches its deadline. weeks.

Max_Duration(1): the duration of the critical path from the last phase to phase 1. weeks.

Max_Duration(2): the duration of the critical path from the last phase to phase 2. weeks.

Max_Duration(3): the duration of the critical path from the last phase to phase 3. weeks.

Max_Duration(4): the duration of the critical path from the last phase to phase 4. weeks.

Max_Duration(5): the duration of the critical path from the last phase to phase 5. weeks.

Max_Headcount(Phase): the most full time developers possible on a single Phase. persons.
Max_Proj_DL_Change: a model control parameter to prevent extreme and unrealistic schedule pressures as the project approaches its deadline. weeks.

Max_Workweek(Phase): Highest possible hours per week each person can spend on project. Set at 140. hours per week per person.

Min_Exp_BW_Prdy(Phase): minimum expected productivity of basework development activities. Primarily used to prevent division by zero or unrealistically high required headcount due to division by a very small number. tasks per person per hour.

Min_Exp_Coord_Prdy(Phase): minimum expected productivity of coordination activities. Primarily used to prevent division by zero or unrealistically high required headcount due to division by a very small number. tasks per person per hour.

Min_Exp_QA_Prdy(Phase): minimum expected productivity of quality assurance activities. Primarily used to prevent division by zero or unrealistically high required headcount due to division by a very small number. tasks per person per hour.

Min_Exp_RW_Prdy(Phase): minimum expected productivity of rework activities. Primarily used to prevent division by zero or unrealistically high required headcount due to division by a very small number. tasks per person per hour.

Min_Headcount(Phase): the fewest number of full time developers possible on a single Phase. persons.

Multiple_Corruption_discoveries(Phase): the number of corrupted tasks having been flawed due to more than one inherited upstream error. This variable is used to correct the number of corrupted tasks for multiple findings of the same corrupted task. dimensionless.

Net_Corrupted_and_Found_Tasks(Phase): the corrected (for multiple findings of flawed tasks) number of tasks found to be corrupted by upstream inherited errors. tasks.

Net_Exper_Gain(Phase): the sum of the experience gained by performing basework and adding new developers to the phase. experiences.

New_Memb_Exper_Gain(Phase): the increase in experience gained by adding new developers to the phase. experiences.

Normal_Workweek(Phase): reference workweek length. Set at 40. hours per week per person.

NotStoppedFlag(Phase): an indicator that a Phase has not been completed. dimensionless.

Not.Done(Phase): a model control parameter to indicate when a phase is active. dimensionless.

OT_Cost(Phase): the overtime costs of a phase. dollars.

Our_Discd_Errors(Phase): the number of tasks which are known to be flawed. tasks.

Our_Discd_Error_density(Phase): the percent of tasks known to require rework which contain errors generated by the focal development Phase. dimensionless.

Our_Errors_Released(Phase): the number of tasks with errors generated within the focal development Phase which have been released to downstream development Phases. tasks.
**Our Undisclosed Errors(Phase):** the number of tasks with errors generated within the focal development Phase which have not been checked for errors. tasks.

**Overtime Premium(Phase):** the percent added to straight time hourly cost to estimate overtime labor cost. Set at 50%. dimensionless.

**Over_Time(Phase):** the number of hours spent over 40 hours per week per developer in a phase. hours.

**Path1:** a switch indicating whether a path through the project network is active. Set to 0 if not used and 1 if used, this switch is used to nullify any inactive paths in schedule and deadline calculations. dimensionless.

**Path12:** a switch indicating whether a path through the project network is active. Set to 0 if not used and 1 if used, this switch is used to nullify any inactive paths in schedule and deadline calculations. dimensionless.

**Path123:** a switch indicating whether a path through the project network is active. Set to 0 if not used and 1 if used, this switch is used to nullify any inactive paths in schedule and deadline calculations. dimensionless.

**Path1234:** a switch indicating whether a path through the project network is active. Set to 0 if not used and 1 if used, this switch is used to nullify any inactive paths in schedule and deadline calculations. dimensionless.

**Path12345:** a switch indicating whether a path through the project network is active. Set to 0 if not used and 1 if used, this switch is used to nullify any inactive paths in schedule and deadline calculations. dimensionless.

**Path1235:** a switch indicating whether a path through the project network is active. Set to 0 if not used and 1 if used, this switch is used to nullify any inactive paths in schedule and deadline calculations. dimensionless.

**Path124:** a switch indicating whether a path through the project network is active. Set to 0 if not used and 1 if used, this switch is used to nullify any inactive paths in schedule and deadline calculations. dimensionless.

**Path1245:** a switch indicating whether a path through the project network is active. Set to 0 if not used and 1 if used, this switch is used to nullify any inactive paths in schedule and deadline calculations. dimensionless.

**Path125:** a switch indicating whether a path through the project network is active. Set to 0 if not used and 1 if used, this switch is used to nullify any inactive paths in schedule and deadline calculations. dimensionless.

**Path13:** a switch indicating whether a path through the project network is active. Set to 0 if not used and 1 if used, this switch is used to nullify any inactive paths in schedule and deadline calculations. dimensionless.

**Path134:** a switch indicating whether a path through the project network is active. Set to 0 if not used and 1 if used, this switch is used to nullify any inactive paths in schedule and deadline calculations. dimensionless.
Path1345: a switch indicating whether a path through the project network is active. Set to 0 if not used and 1 if used, this switch is used to nullify any inactive paths in schedule and deadline calculations. dimensionless.

Path135: a switch indicating whether a path through the project network is active. Set to 0 if not used and 1 if used, this switch is used to nullify any inactive paths in schedule and deadline calculations. dimensionless.

Path14: a switch indicating whether a path through the project network is active. Set to 0 if not used and 1 if used, this switch is used to nullify any inactive paths in schedule and deadline calculations. dimensionless.

Path145: a switch indicating whether a path through the project network is active. Set to 0 if not used and 1 if used, this switch is used to nullify any inactive paths in schedule and deadline calculations. dimensionless.

Path15: a switch indicating whether a path through the project network is active. Set to 0 if not used and 1 if used, this switch is used to nullify any inactive paths in schedule and deadline calculations. dimensionless.

Path2: a switch indicating whether a path through the project network is active. Set to 0 if not used and 1 if used, this switch is used to nullify any inactive paths in schedule and deadline calculations. dimensionless.

Path23: a switch indicating whether a path through the project network is active. Set to 0 if not used and 1 if used, this switch is used to nullify any inactive paths in schedule and deadline calculations. dimensionless.

Path234: a switch indicating whether a path through the project network is active. Set to 0 if not used and 1 if used, this switch is used to nullify any inactive paths in schedule and deadline calculations. dimensionless.

Path2345: a switch indicating whether a path through the project network is active. Set to 0 if not used and 1 if used, this switch is used to nullify any inactive paths in schedule and deadline calculations. dimensionless.

Path235: a switch indicating whether a path through the project network is active. Set to 0 if not used and 1 if used, this switch is used to nullify any inactive paths in schedule and deadline calculations. dimensionless.

Path24: a switch indicating whether a path through the project network is active. Set to 0 if not used and 1 if used, this switch is used to nullify any inactive paths in schedule and deadline calculations. dimensionless.

Path245: a switch indicating whether a path through the project network is active. Set to 0 if not used and 1 if used, this switch is used to nullify any inactive paths in schedule and deadline calculations. dimensionless.

Path25: a switch indicating whether a path through the project network is active. Set to 0 if not used and 1 if used, this switch is used to nullify any inactive paths in schedule and deadline calculations. dimensionless.
Path3: a switch indicating whether a path through the project network is active. Set to 0 if not used and 1 if used, this switch is used to nullify any inactive paths in schedule and deadline calculations. dimensionless.

Path34: a switch indicating whether a path through the project network is active. Set to 0 if not used and 1 if used, this switch is used to nullify any inactive paths in schedule and deadline calculations. dimensionless.

Path345: a switch indicating whether a path through the project network is active. Set to 0 if not used and 1 if used, this switch is used to nullify any inactive paths in schedule and deadline calculations. dimensionless.

Path35: a switch indicating whether a path through the project network is active. Set to 0 if not used and 1 if used, this switch is used to nullify any inactive paths in schedule and deadline calculations. dimensionless.

Path4: a switch indicating whether a path through the project network is active. Set to 0 if not used and 1 if used, this switch is used to nullify any inactive paths in schedule and deadline calculations. dimensionless.

Path45: a switch indicating whether a path through the project network is active. Set to 0 if not used and 1 if used, this switch is used to nullify any inactive paths in schedule and deadline calculations. dimensionless.

Path5: a switch indicating whether a path through the project network is active. Set to 0 if not used and 1 if used, this switch is used to nullify any inactive paths in schedule and deadline calculations. dimensionless.

Percent_hourly_Labor(Phase): the portion of the total labor hours which are subject to overtime pay. dimensionless.

percent_Multiple_Corruption_discoveries(Phase): the probability of a corrupted task having been flawed due to more than one inherited upstream error. This variable is used to correct the number of corrupted tasks for multiple findings of the same corrupted task. dimensionless.

\$\text{Productivity}\$ the project quality goal described in terms of the amount of rework. dimensionless.

Percent_Tasks_in_Rework_Que(Phase): the percent of tasks in a Phase which are known to require rework. This variable is used to find the perceived quality for comparison with the quality goal.

PhaseNo(1): a model control parameter to identify a phase of the project. dimensionless.

PhaseNo(2): a model control parameter to identify a phase of the project. dimensionless.

PhaseNo(3): a model control parameter to identify a phase of the project. dimensionless.

PhaseNo(4): a model control parameter to identify a phase of the project. dimensionless.

PhaseNo(5): a model control parameter to identify a phase of the project. dimensionless.

Phase_Cost_to_Date(Phase): the accumulation of all costs incurred by a phase up to the current simulation time. This value is used with the forecast of costs to estimate total costs of the phase. dollars.
Phase_Deadline(Phase):  the date by which the Phase must be completed (all tasks correct and released) to have the project completed by the current Project Deadline. This time is found by subtracting the longest duration (determined by the paths of Phase dependencies from the Phase to the project completion) from the project deadline. weeks from start.

Press_for_BW(Phase):  the pressure on developers to allocate available labor to the basework activity. dimensionless.

Press_for_Coord(Phase):  the pressure on developers to allocate available labor to the coordination activity. dimensionless.

Press_for_QA(Phase):  the pressure on developers to allocate available labor to the quality assurance activity. dimensionless.

Press_for_RW(Phase):  the pressure on developers to allocate available labor to the rework activity. dimensionless.

prob_err_gen_fr_Task_Complexity(Phase)= a table function which relates the complexity of the task to the probability of generating an error in a task being performed. dimensionless.

Prob_finding_our_error_if_exists(Phase):  a table function which relates levels of quality assurance, quality of practice, and complexity to the probability of finding internally generated errors. dimensionless.

prob_find_Up_error_if_exists(Phase):  the probability of finding an error in a focal Phase task which was caused by an error inherited from an upstream development Phase if the error exists in the task. dimensionless.

prob_of_err_gen_by_QofP(Phase):  a table function which relates how well the quality of practice needs are being met to the probability of generating an error in a task being performed. dimensionless.

prob_of_no_err_gen_from_effects(Phase):  the chance that a task is flawed due to the effects of Task Complexity and Quality of Practice. dimensionless.

prob_Task_Corrupted_and_found(up,Phase):  the probability that a task contains an error due to an inherited error and that the induced error was found by the quality assurance effort. dimensionless.

prob_Task_flawed(Phase):  the chance that a task is flawed due the effects of Task Complexity, Quality of Practice, and the basic difficulty of performing the task. dimensionless.

prob_Task_flawed_and_FOUND(Phase):  the chances that a task both contains an error and the quality assurance process found the error. dimensionless.

Project_Cost_to_Date:  the sum of the costs of all active phases incurred to the current simulation time. dollars.

Project_Deadline:  the current planned completion date of all Phases. weeks from start.

Project_Errors_Released:  the sum of the flawed tasks released by all active phases. tasks.

Project_Quality_Goal:  the percent of tasks which the developers want to be released without flaws. dimensionless.
**Proj_Budget:** the cost allocated for the project. This is compared with forecasted costs to establish the budget status. dollars.

**Proj_Quality_Gap:** the difference between the objective and actual rework percentages. dimensionless.

**Proj_Sched_Press:** the ratio of the time estimated to be required to complete the project to the time available to complete the project. dimensionless.

**Proj_Sched_Slip:** the change in the project deadline in response to project schedule pressure. weeks.

**Proj_Tasks_Compl_and_Rel:** a record keeping parameter which is the sum of the Completed, not Checked and Tasks Released stocks for all phases. tasks.

**QA_for_Find_Error(Phase):** the measure of the adequacy of quality assurance efforts in a Phase, defined by the ratio of the required to actual quality assurance labor. dimensionless.

**QA_inspection_rate(Phase):** the rate at which completed tasks are inspected for errors. tasks per week.

**QA_Labor(Phase):** the amount of labor applied to quality assurance activities in a focal Phase. person-hours.

**QA_Labor_Delay(Phase):** the time between the need for quality assurance labor as reflected in the pressure for development activities and when labor allocation shifts in response to that pressure. weeks.

**QA_Labor_Limit(Phase):** the maximum quality assurance rate allowed by the availability and effectiveness of resources for quality assurance. tasks per week.

**QA_Labor_Required(Phase):** the number of full time experienced, rested developers required to complete the currently available quality assurance work based on the perceived productivity. developers.

**QA_Min_Task_Duration(Phase):** the minimum time required to check a single task for errors, assuming no resource or available-work constraints. weeks.

**QA_Prdctvty_Avg_Time(Phase):** the smoothing time for perceived quality assurance productivity based on the assumption that developers average instantaneous productivity to predict labor needs. weeks.

**QA_Prdctvty_Report_Time(Phase):** the delay in reporting the actual productivity of quality assurance activities. weeks.

**QA_Priority(Phase):** the importance of applying available labor to quality assurance activities in relation to the importance of basework, rework, or coordination activities. dimensionless.

**QA_Process_Limit(Phase):** the maximum rate of checking tasks for errors possible based upon the quality assurance process. tasks per hour.

**QA_Prdctvty_Report_Time(Phase):** the time delay between actual and use of reported productivity. This variable is fairly long since formal means of productivity reporting are not used and team members effectively report productivity through experiencing the progress and effort of the project. weeks.

**QA_STATUS:** the measure of the adequacy of quality assurance efforts in a Phase, defined by the ratio of the required to actual quality assurance labor. dimensionless.
**QoF_Error_Disc_effect(Phase):** a table function which relates the quality of practice to the probability of finding an internally generated error. dimensionless.

**Quality_Goal(Phase):** the percent of tasks without defects which is the objective of a phase. percent defects.

**Quality_Goal_Adjust_Time(Phase):** the time over which the development team of a Phase allows its quality goal to move to the current quality performance. This is rather long, reflecting the strong history and current priority on producing error-free products to customers and downstream Phases. weeks.

**Quality_Goal_Switch:** a model control parameter which engages the quality feedback loops. dimensionless.

**Quality_of_Practice(Phase):** the relative performance of the work of developers without error, as influenced by project conditions such as schedule pressure and fatigue. dimensionless

**Qual_Gap_effect_on_Coord_Import:** a table function which describes the impact of the project's quality gap on the priority of doing coordination. Coordination priority generally increases with increasing quality gap. dimensionless.

**Qual_Gap_effect_on_QARW_priority:** a table function which describes the impact of the project's quality gap on the priority of doing quality assurance and rework. These priorities generally increase with increasing quality gap. dimensionless.

**Receive_Our_Errors_fr_Dn(Phase):** the rate at which tasks which are flawed are returned to the focal development Phase for rework from downstream development Phases. tasks per week.

**Ref_BW_Prdtvty(Phase):** benchmark productivity of labor at basework development activities. tasks per person per hour.

**Ref_BW_Tasks_per_Hour(Phase):** benchmark productivity of process at basework development activities. tasks per person per hour.

**Ref_Complexity(Phase):** benchmark complexity of project for comparison of different projects. Impacts error generation and discovery. dimensionless.

**Ref_Coord_Prdtvty(Phase):** benchmark productivity of labor at coordination activities. tasks per person per hour.

**Ref_Exper(Phase):** benchmark amount of experience for finding effect of cumulative experience on productivity. Experience is measured in units similar to tasks, with the performance of each task generating one more unit of experience. experiences.

**Ref_QA_Prdtvty(Phase):** benchmark productivity of labor at quality assurance activities. tasks per person per hour.

**Ref_Qual_of_Practice(Phase):** a benchmark quality of practice for Phase work. dimensionless.

**Ref_RW_Prdtvty(Phase):** benchmark productivity of labor at quality assurance activities. tasks per person per hour.
Release Any Errors Flag(Phase): an indicator if any internally generated errors were released from a Phase. weeks that an error was released.

Release Any Errors Flow(Phase): the rate for the indicator if any internally generated errors were released from a Phase. dimensionless.

Release Any Tasks Flag(Phase): an indicator if any tasks were released from a Phase. weeks that a task was released.

Release Any Tasks Flow(Phase): the rate for the indicator if any tasks were released from a Phase. dimensionless.

Release Errors(Phase): the rate at which errors generated within the focal development Phase are released to downstream Phases. tasks per week.

Release Errors from Hold(Phase): the rate at which errors generated within a phase are released to downstream as a group. tasks per week.

Release Hold Avg(Phase): the rolling average of the number of tasks being held for group release. tasks.

Release Hold Avg Stability: a measure of the change in the holding stock before release as a group. This is used to simulate the developers waiting until task completion rates become small (stable average), indicting that they have fixed errors. dimensionless.

Release Hold Avg Time(Phase): the time over which the tasks being held for group release are averaged. weeks.

Release Tasks(Phase): The rate at which tasks which have been checked for errors are released by the focal development Phase to downstream development Phases. tasks per week.

Release Tasks from Hold(Phase): the rate of group release of tasks which have been accumulated, based upon the stable average of the number of tasks waiting for release. tasks per week.

Release Trigger(Phase): the indicator of the adequate accumulation of tasks for group release, based upon the stable average of the number of tasks waiting for release. dimensionless.

Release Trigger Sensativity(Phase): the level of stability required before the held tasks are released. dimensionless.

Release Trigger Task Gate(Phase): the release trigger switch. dimensionless.

Rel Task_error_density(Phase): the percent of tasks released to downstream Phases which contain errors. dimensionless.

Report BW Prdctvty Time(Phase): the time delay between actual and use of reported productivity. This variable is fairly long since formal means of productivity reporting are not used and team members effectively report productivity through experiencing the progress and effort of the project. weeks.

Report Coord Prdctvty time(Phase): the time delay between actual and use of reported productivity. This variable is fairly long since formal means of productivity reporting are not used and team members effectively report productivity through experiencing the progress and effort of the project. weeks.
Required_Headcount(Phase): the number of persons required to complete the Phase by the Phase deadline based upon the estimated hours required to finish and current headcount and normal workweek. persons.

Required_Personweeks(Phase): the number of labor hours required to complete the phase. developers.

Resistance_to_Sched_Slip: the ratio of required to available time at which the project deadline is slipped. dimensionless.

ResourseSwitch(Phase): switch enabling and disabling the labor portions of the model. dimensionless.

Rework_Any_Flag(Phase): an indicator if any tasks were reworked in a Phase. weeks that a task was reworked.

Rework_Any_Flow(Phase): the rate for the indicator if any tasks were reworked in a Phase. dimensionless.

Rework(Phase): The rate at which tasks which are known to require rework are corrected. New errors may be generated in these tasks while they are being reworked. tasks per week.

RW_due_to_Corrupted_tasks(Phase): tasks known to require rework which were flawed by an upstream development Phase and in which the flaws were discovered by quality assurance efforts within the focal development Phase. tasks per week.

RW_due_to_InPhase_QA(Phase): tasks known to require rework which were flawed by the focal development Phase and in which the flaws were discovered by quality assurance efforts within the focal development Phase. tasks per week.

RW_due_to_Dwnstrm_QA(Phase): tasks known to require rework which were flawed by the focal development Phase and in which the flaws were discovered by quality assurance efforts within a downstream development Phase. tasks per week.

RW_Labor(Phase): the time between the need for rework labor as reflected in the pressure for development activities and when labor allocation shifts in response to that pressure. weeks.

RW_Labor_Delay(Phase): the time between the need for rework labor as reflected in the pressure for development activities and when labor allocation shifts in response to that pressure. weeks.

RW_Labor_Limit(Phase): the maximum rework rate allowed by the availability and effectiveness of resources for rework. tasks per week.

RW_Labor_Required(Phase): the number of full time experienced, rested developers required to complete the currently available rework based on the perceived productivity. developers.

RW_Min_Task_Duration(Phase): the minimum time required to complete a single task when the task is done subsequent to the first time, assuming no resource or available-work constraints. weeks.

RW_Prdctvty_Avg_Time(Phase): the smoothing time for perceived rework productivity based on the assumption that developers average instantaneous productivity to predict labor needs. weeks.

RW_Priority(Phase): the importance of applying available labor to rework in relation to the importance of basework, quality assurance or coordination activities. dimensionless.
**RW_Process_Limit(Phase):** the maximum rate of reworking tasks which are known to be flawed possible based upon the rework process. tasks per hour.

**Sched_BW_Press_effect(Phase):** a table function which relates the amount of schedule pressure to the priority given to spending available time on basework development activities. dimensionless.

**Sched_Error_Disc_effect(Phase):** a table function which relates the Schedule Pressure to the probability of discovering an error if the error exists. dimensionless.

**Sched_Pressure(Phase):** The ratio of the time required to complete a Phase to the time to the Phase deadline. If schedule pressure is < 1.00 (e.g., 0), Quality of Practice drops and not all errors will be caught. Set schedule pressure to 1.00 for "no schedule pressure" testing. dimensionless.

**Sched_Qual_of_Prac_effect(Phase):** a table function which relates Schedule Pressure to the Quality of Practice. dimensionless.

**Sched_Workweek_effect(Phase):** a table function which relates the level of schedule pressure to the number of hours worked per week. dimensionless.

**StartFlag(Phase):** a indicator that a Phase has begun. dimensionless.

**Start_Time(Phase):** time Phase starts, defined as when basework begins. weeks from start of project.

**StoppedFlag(Phase):** a model control parameter to indicate when a phase has ended. dimensionless.

**Straight_Cost(Phase):** the cost incurred due to non overtime work. dollars.

**Straight_Time(Phase):** the hours worked within the limit of 40 hours per week per developer. hours.

**Sum_of_Tasks(Phase):** the total number of tasks that have been completed at least once in a Phase, i.e. the completed, known rework, released, and hold for release tasks. tasks.

**TaskListScale(up,down):** the scaling factor which relates the relative sizes of two Phases. dimensionless

**Tasks_Completed(Phase):** the number of tasks in a development Phase which have been initially finished but have not been checked for errors. tasks.

**Tasks_Compl_and_Holding(Phase):** the number of tasks which have been completed and not yet checked for errors or have been completed, checked for errors, and are being held for group release to downstream development Phases. tasks.

**Tasks_Compl_and_Rel(Phase):** the number of tasks which have been completed and not yet checked for errors or have been completed, checked for errors, and released to downstream development Phases. tasks.

**Tasks_Corrupted_and_Found(up,Phase):** the number of tasks that contain an error due to an inherited error and the induced error was found by the quality assurance effort. tasks.

**Tasks_Released(Phase):** the number of tasks in a development Phase which have been completed, checked for errors, and released to downstream development Phases. tasks.

**Tasks_to_Release_Hold(Phase):** the rate of tasks checked and believed to have no errors to accumulate for release as a group. tasks per week.
**Task_Complex_effect_on_Err_disc(Phase)**: a table function which relates the complexity of the task to the probability of finding an error generated by the focal development Phase. dimensionless.

**Task_List(Phase)**: the number of "tasks" or pieces of atomic work which must be completed correctly for a Phase to be finished. A task is small enough that it is completely correct or completely incorrect. number of tasks.

**Test_Input_1(Phase)**: the switch enabling and disabling errors and rework in the focal Phase. dimensionless.

**Test_Input_2(Phase)**: the switch which selects release direct from completed tasks or holding of completed tasks for clustered release of tasks. This is used to model the aggregation of design products and their release together to make masks. dimensionless.

**Test_Input_3(Phase)**: a model control parameter which is not used. dimensionless.

**Time_Required(Phase)**: the time needed to complete a Phase, based upon the required person weeks and current headcount. weeks.

**Time_spent_to_Date(Phase)**: the number of weeks between the current time (or the Phase deadline if Phase is completed) and the week when the Phase began. weeks.

**Time_to_Avg_Exp_Compl_Time(Phase)**: a short smoothing parameter to prevent extreme fluctuations in expected completion times and thereby extreme movement of the project deadline. weeks.

**Time_to_Deadline(Phase)**: the number of weeks from the current time to the Phase Deadline. weeks.

**Time_Phase_Active(Phase)**: the flag signaling that a Phase in active (begun but not completed). dimensionless.

**Total_Err_disc_by_Dn(Phase)**: the sum of errors passed to downstream Phases by the focal Phase which are returned to the focal Phase for rework. tasks.

**Total_Hdct**: the sum of the active headcount of all phases. developers.

**Total_Labor_Required(Phase)**: the sum of the labor required by all four development activities. developer hours.

**Total_Pressure_for_Activities(Phase)**: the sum of the pressures on developers to allocate labor to the four development activities. dimensionless.

**Tot_Exp_Costs**: the sum of the total expected cost of all phases in the project. dollars.

**Tot_Tasks_Avail(Phase)**: the number of tasks which are available to be completed the first time (basework). tasks.

**Up_Flawed_Tasks_found(up,Phase)**: the number of tasks in each upstream Phase which were responsible for the errors in the focal Phase which were caused by inherited errors. This variable is used to pass error information back to upstream Phases. tasks.

**Wait_Time(Phase)**: time before Phase starts. weeks
Workweek(Phase): the average number of hours worked in each week by each developer in a phase. Hours per week.

Wkwk_Avg_Time(Phase): the time over which the average workweek is calculated. This variable is used to find fatigue levels. It represents the time it takes for extended overtime work to impact workers. Set at 4 weeks.

Wt_to_Reported_BW_Prdctvty(Phase): the percent of influence given to the reported productivity of basework development activities. This variable is balanced by the historical productivity assumptions held by team members. Set at 50%. Dimensionless.
Appendix 4.1

Signal Processing Model

// SIGNAL PROCESSING MODEL

// x is the vector of state variables
// u is the input signal (a function of t)
// X1 = COLUMN 2 IS THE STOCK OF COMPLETED, NOT CHECKED TASKS
// X2 = COLUMN 3 IS THE STOCK OF KNOWN REWORK
// X3 = COLUMN 4 IS THE STOCK OF TASKS RELEASED

// Start simulation time
// End simulation time
// Time increment (dt)

tstart=1;
tend=100;
tincrement=0.1;

// FUNCTION CALCULATING CHANGE IN OUTPUT = XDOT

xdot=function(t,x)
{

// Declare variables

global (Frequency);
global (AmpIn);
local (Output);
global (Tau1);
global (Tau2);
global (IFR);
global (Par);
Output=zeros(3,1);
Freq=Frequency;
// INPUT SIGNAL
AmpIn=3; // Amplitude of Input Signal
theta=0; // Phase shift of frequency input signal
Beta=(t*Freq*2*3.1415962) + theta;
u=AmpIn+ AmpIn*sin(Beta); // SINE WAVE INPUT

Tau1=Par; // Inspection Time Constant, to be varied for 3d plot
Tau2=8;
IFR=0.25; // Inspection Failure Fraction
A=[-1/Tau1, 1/Tau2, 0; IFR/Tau1, -1/Tau2, 0; (1-IFR)/Tau1, 0, 0];
B=[1,0,0];
Output=A*x + B*u;
return Output;
}

// LOOP SIMULATION THROUGH PARAMETER VALUES
MinPar=1; // Starting parameter value
MaxPar=10; // Ending parameter value
FrequencyIterations=21; // Number of parameter values simulated
Data=zeros(FrequencyIterations,5);
MaxFrequency=10;
Y=zeros(MaxPar-MinPar+1,1);
X=zeros(FrequencyIterations,1);
Z=zeros(size(X)[1],size(Y)[1]);

for(i in MinPar:MaxPar) // Set up output matrix
{
  Y[i]=i;
  printf("Par=%d",i);
  Par=i;

  for(f in 1:FrequencyIterations)
  {
    printf("i = %d",i);
    Frequency=(f/FrequencyIterations)*MaxFrequency;
    X[f]=Frequency;
    printf("Frequency=%d",Frequency);
    ystart=[0,0,0];
    y=ode(xdot, tstart, tend, ystart, tincrement, , ); // integration function call
    LargerY=zeros(size(y)[1],5);
    LargerY[:,1]=y[:,1];
LargerY[2]=y[2]*(1-IFR)*(1/Tau1);  //ReleaseRate vector
LargerY[3]=y[2]*(IFR/Tau1);         // REWORK DUE TO QA FLOW
LargerY[4]=y[3]/Tau2;               // REWORK FLOW

// OPTIONAL CALCULATION OF SYSTEM GAIN FOR EACH SIMULATION
MaxReleaseRate=max(LargerY[2]);     //End of Transient response
BeginSteadyState=70;
MinReleaseRate=min(LargerY[BeginSteadyState,.size(LargerY)[1];2]);
AmpOut=MaxReleaseRate-MinReleaseRate;
Gain=AmpOut/AmpIn;
printf("Gain = %.2f\n",Gain);

Data[f;1]=i;
Data[f;2]=Frequency;
Data[f;3]=Gain;
// printf("\nInto Data");
}
// ==============
for (no in 1:.size(X)[1])
{Z[no;i]=Data[no;3];}

// PLOT OUTPUT

// 3D PLOT OF SYSTEM GAIN
pstart(1,1,"Mac");
ptitle("System Gain ");
xlabel("Frequency");
ylabel("T1");
zlabel("System Gain");
plmesh(<x=X;y=Y ;z=Z>);

pstart(1,1,"li_hpgl");
ptitle("System Gain");
xlabel("Frequency");
ylabel("T1");
zlabel("System Gain");
plmesh(<x=X;y=Y ;z=Z>);
// PLOT SYSTEM GAIN ON SCREEN
pstart(1,1,"Mac");
ptitle("Frequency versus System Gain");
xlabel ("Frequency");
ylabel ("Gain");
plot([[Data[2],Data[3]]]);

// PLOT EXAMPLE SIMULATION ON SCREEN
pstart(1,1,"Mac");
ptitle("Highest Frequency Task Release Rate");
xlabel ("Time");
ylabel ("Development Tasks per Time Unit");
plot([LargerY[1],LargerY[2]]);

// PLOT EXAMPLE SIMULATION ON PRINTER
pstart(1,1,"lj_hpgl");
ptitle("Highest Frequency Task Release Rate");
xlabel ("Time");
ylabel ("Development Tasks per Time Unit");
plot([LargerY[1],LargerY[2]]);

// PLOT EXAMPLE SIMULATION ON SCREEN
pstart(1,1,"Mac");
ptitle("Highest Frequency Rework due to QA");
xlabel ("Time");
ylabel ("Development Tasks per Time Unit");
plot([LargerY[1],LargerY[3]]);

// PLOT EXAMPLE SIMULATION ON PRINTER
pstart(1,1,"lj_hpgl");
ptitle("High Frequency Rework due to QA");
xlabel ("Time");
ylabel ("Development Tasks per Time Unit");
plot([LargerY[1],LargerY[3]]);
// PLOT EXAMPLE SIMULATION ON SCREEN
pstart(1,1,"Mac");
ptitle("Highest Frequency Rework Flow");
xlabel ("Time");
ylabel ("Development Tasks per Time Unit");
plot([LargerY[1],LargerY[4]]);

// PLOT EXAMPLE SIMULATION ON PRINTER
pstart(1,1,"lj_hpgl");
ptitle("High Frequency Rework Flow");
xlabel ("Time");
ylabel ("Development Tasks per Time Unit");
plot([LargerY[1],LargerY[4]]);
Appendix 5.1

Parameter Estimates for Model Calibration

See Appendix 3.1 Model Equations for a complete listing of the model.
See Appendix 3.2 Model Variables for definitions of all model variables.

Single Phase Model Parameter Estimates

PROCESS PARAMETERS
BW_Min_Task_duration(1)=2 hours
RW_Min_Task_Duration(1)=1 hours
QA_Min_Task_Duration(1)=0.75 hours
Internal Precedence Relationship: (dimensionless)
Values of (Percent Completed and Released, Percent Available for Completion) are: (0,0.01) (0.1,0.21)
90.2,0.31) (0.3,0.41) (0.4,0.51) (0.5,0.61) (0.6,0.71) (0.7,0.81) (0.8,0.91) (0.9,1.00) (1,1.00)
Release_Trigger_Sensativity(Phase)=0.1 (dimensionless)
Release_Hold_Avg_Time(Phase)=1.05 weeks

SCOPE PARAMETERS
Task_List(1)=445

TARGET PARAMETERS
Schedule
Initial_Proj_Deadline=25 (weeks from phase start)
Time_to_Avg_Exp_Compl_Time(Phase)=1 weeks
Resistance_to_Sched_Slip=2 (dimensionless)

Quality
Quality_Goal_Adjust_Time(Phase)=24 weeks
Project_Quality_Goal=1.00 (dimensionless percent defects)
Initial_Quality_Goal(Phase)=.00 (dimensionless percent defects)
Basic_prob_flawed_Task(Phase)=0.85 (dimensionless)
Complexity(Phase)=10 (dimensionless)

Cost
Budget_Switch=0 (dimensionless)

RESOURCE PARAMETERS
Gross Labor
Inital_Headcount(Phase)=0.50 (persons)
Max_Headcount(Phase)=2 (persons)
Headcount_Adjustment_Time(Phase)=8 (1/weeks)
HdctJumpStartTime(Phase)=11 (weeks from start)
HdctJumpStopTime(Phase)=14 (weeks from start)
Max_Workweek(Phase)=140 (hours per week)
Normal_Workweek(Phase)=40 (hours per week)
Wrkwk_Avg_Time(Phase)=4 (1/weeks)

Labor Allocation
BW_Priority(Phase)=3 (dimensionless)
RW_Priority(Phase)=1 (dimensionless)
QA_Priority(Phase)=1 (dimensionless)
BW_Labor_Delay(Phase)=1.5 (1/weeks)
RW_Labor_Delay(Phase)=1 (1/weeks)
QA_Labor_Delay(Phase)=7.75 (1/weeks)

Experience
Exper_Assim_Time(Phase)=1 (1/weeks)
Avg_New_member_Exper(Phase)=6 (experience units)

Productivity
Ref_BW_Prdctvty(Phase)=2 (tasks per person per hour)
Ref_RW_Prdctvty(Phase)=1 (tasks per person per hour)
Ref_QA_Prdctvty(Phase)=1.75 (tasks per person per hour)
BW_Prdctvty_Avg_Time(Phase)=1 (1/weeks)
RW_Prdctvty_Avg_Time(Phase)=1 (1/weeks)
QA_Prdctvty_Avg_Time(Phase)=1 (1/weeks)
Adjust_Expect_BW_Prdctvty_Time(Phase)=1 (1/weeks)
Change_Expected_QA_Prdctvty_Time(Phase)=1 (1/weeks)
BW_Prdctvty_Influences_Time(Phase)=1 (1/weeks)
Ch_Expect_Coord_Prdctvty_time(Phase)=1 (1/weeks)
QA_Prdctvty_Report_Time(Phase)=1 (1/weeks)
Report_BW_Prdctvty_Time(Phase)=1 (1/weeks)
Wt_to_Current_BW_Prdctvty(Phase)=1.00 (dimensionless)

Model Control Parameters

LastPhase=1
PhaseNo(1)=1
ResourceSwitch(Phase)=1
Test_Input_1(Phase)=1
Test_Input_2(1)=1
Test_Input_3(Phase)=1
Quality_Goal_Switch=1
Deadline_Switch=1
alpha(Phase)=100
QA_STATUS(Phase)=1
CloseEnough=0.01
Min_Exp_RW_Prdy(Phase)=0.1
Min_Exp_BW_Prdy(Phase)=0.1
Min_Exp_QA_Prdy(Phase)=0.1
Min_Headcount(Phase)=0.001
Max_Proj_DL_Change=100
Max_DL_Change(Phase)=100
COORD_STATUS(test)(Phase)=0.5
Coord_Min_duration(1)=1
Coord_Labor_Delay(Phase)=1
Ref_Coord_Prdctvty(Phase)=1
Min_Exp_Coord_Prdy(Phase)=0.1
Avg_Act_Coord_Prdctvty_Time(Phase)=1
Report_Coord_Prdctvty_time(Phase)=1 (1/weeks)
Coord_Priority(Phase)=1
Coord_Prdctvty_Avg_Time(Phase)=1
Proj_Budget=500000
Overtime_Premium(Phase)=0.50
CostMarkup(Phase)=2
Avg_Straight_Pay(Phase)=15
Percent_hourly_Labor(Phase)=0.00
Ref_Qual_of_Practice(Phase)=5
Ref_Exper(Phase)=50
Ref_Complexity(Phase)=100
Avg_Act_BW_Prdctvty_Time(Phase)=1 (1/weeks)
Avg_Act_RW_Prdctvty_Time(Phase)=1 (1/weeks)
Avg_Act_QA_Prdctvty_Time(Phase)=1 (1/weeks)

Multiple Phase Model Parameter Estimates

Notes:
Phases are identified by the integer in parenthesis at the end of the parameter name.
Phase 1 is Product Definition.
Phase 2 is Design.
Phase 3 is Prototype Testing.
Phase 4 is Reliability/Quality Control.
Phase 5 is not used.
"Phase" instead of a phase number or no phase identifier indicates that all phases use the value given.

PROCESS PARAMETERS
Basic_prob_flawed_Task(1)=0.5
Basic_prob_flawed_Task(2)=0.3
Basic_prob_flawed_Task(3)=0.05
Basic_prob_flawed_Task(4)=0.05
Basic_prob_flawed_Task(5)=0
BW_Min_Task_duration(1)=5
BW_Min_Task_duration(2)=2
BW_Min_Task_duration(3)=6
BW_Min_Task_duration(4)=2
BW_Min_Task_duration(5)=2
Coord_Min_duration(1)=1
Coord_Min_duration(2)=1
Coord_Min_duration(3)=1
Coord_Min_duration(4)=1
Coord_Min_duration(5)=1
Dependency(1,1)=0
Dependency(1,2)=1
Dependency(1,3)=1
Dependency(1,4)=0
Dependency(1,5)=0
Dependency(2,1)=0
Dependency(2,2)=0
Dependency(2,3)=1
Dependency(2,4)=1
Dependency(2,5)=0
Dependency(3,1)=0
Dependency(3,2)=0
Dependency(3,3)=0
Dependency(3,4)=1
Dependency(3,5)=0
Dependency(4,1)=0
Dependency(4,2)=0
Dependency(4,3)=0
Dependency(4,4)=0
Dependency(4,5)=0
Dependency(5,1)=0
Dependency(5,2)=0
Dependency(5,3)=0
Dependency(5,4)=0
Dependency(5,5)=0
QA_Min_Task_Duration(1)=2
QA_Min_Task_Duration(2)=2
QA_Min_Task_Duration(3)=0.5
QA_Min_Task_Duration(4)=1
QA_Min_Task_Duration(5)=1
Release_Hold_Avg_Time(Phase)=1.06
Release_Trigger_Sensativity(1)=0.6
Release_Trigger_Sensativity(2)=0.05
Release_Trigger_Sensativity(3)=0.6
Release_Trigger_Sensativity(4)=0.6
Release_Trigger_Sensativity(5)=0.6
RW_Min_Task_Duration(1)=3
RW_Min_Task_Duration(2)=0.5
RW_Min_Task_Duration(3)=0.5
RW_Min_Task_Duration(4)=1
RW_Min_Task_Duration(5)=1
Internal Precedence Relationships:
T5(*,1)=0.01/0.15/0.3/0.60/0.80/0.90/0.95/1/1/1/1.00 * Product Definition
T5(*,2)=0.01/0.15/0.40/0.5/0.65/0.75/0.85/0.95/0.97/1.00/1.00 * Design
T5(*,3)=0.4/0.5/0.6/0.7/0.8/0.9/0.95/1/1/1/1.00 * Prototype Testing
T5(*,4)=1/1/1/1/1/1/1/1/1/1.00/1.00 * Reliability/Quality Control
T5(*.5)=0/0/0/0/0/0/0/0/0/0/0/0/0/0 *Closed Gate for Phase 5
External Precedence Relationships:
T6(*,1,1)=0.00/0.00/0.00/0.00/0.00/0.00/0.00/0.00/0.00/0.00/0.00/1.00 * NA
T6(*,1,2)=0.00/0.1/0.25/0.5/0.65/0.8/0.9/0.95/0.97/1.00/1.0 * Product Definition to Design
T6(*,1,3)=0.00/0.00/0.00/0.00/0.00/0.00/0.00/0.00/0.00/0.00/0.00/1.00 * Product Definition to Prototype Testing
T6(*,2,1)=0.00/0.00/0.00/0.00/0.00/0.00/0.00/0.00/0.00/0.00/0.00/1.00 * NA
T6(*,2,2)=0.00/0.00/0.00/0.00/0.00/0.00/0.00/0.00/0.00/0.00/0.00/1.00 * NA
T6(*,2,3)=0.01/0.05/0.2/0.4/0.6/0.8/0.9/1.0/1.1/1.1/1.1/1.00 * Design to Prototype Testing
T6(*,2,4)=0.00/0.00/0.00/0.00/0.00/0.00/0.00/0.00/0.00/0.00/0.00/1.00 * Design to Reliability/Quality Control
T6(*,3,1)=0.00/0.00/0.00/0.00/0.00/0.00/0.00/0.00/0.00/0.00/0.00/1.00 * NA
T6(*,3,2)=0.00/0.00/0.00/0.00/0.00/0.00/0.00/0.00/0.00/0.00/0.00/1.00 * NA
T6(*,3,3)=0.00/0.00/0.00/0.00/0.00/0.00/0.00/0.00/0.00/0.00/0.00/1.00 * NA
T6(*,3,4)=0.00/0.00/0.00/0.00/0.00/0.00/0.00/0.00/0.00/0.00/0.00/1.00 * Prototype Testing to Reliability/Quality Control
T6(*,4,1)=0.00/0.00/0.00/0.00/0.00/0.00/0.00/0.00/0.00/0.00/0.00/1.00 * NA
T6(*,4,2)=0.00/0.00/0.00/0.00/0.00/0.00/0.00/0.00/0.00/0.00/0.00/1.00 * NA
T6(*,4,3)=0.00/0.00/0.00/0.00/0.00/0.00/0.00/0.00/0.00/0.00/0.00/1.00 * NA
T6(*,4,4)=0.00/0.00/0.00/0.00/0.00/0.00/0.00/0.00/0.00/0.00/0.00/1.00 * NA
T6(*,5,1)=0.00/0.00/0.00/0.00/0.00/0.00/0.00/0.00/0.00/0.00/0.00/1.00 * NA
T6(*,5,2)=0.00/0.00/0.00/0.00/0.00/0.00/0.00/0.00/0.00/0.00/0.00/1.00 * NA
T6(*,5,3)=0.00/0.00/0.00/0.00/0.00/0.00/0.00/0.00/0.00/0.00/0.00/1.00 * NA
T6(*,5,4)=0.00/0.00/0.00/0.00/0.00/0.00/0.00/0.00/0.00/0.00/0.00/1.00 * NA
T6(*,5,5)=0.00/0.00/0.00/0.00/0.00/0.00/0.00/0.00/0.00/0.00/0.00/1.00 * NA
Test_Input_2(1)=0 * RELEASE"DUMP" SWITCH
Test_Input_2(2)=1 * RELEASE"DUMP" SWITCH
Test_Input_2(3)=0 * RELEASE"DUMP" SWITCH
Test_Input_2(4)=0 * RELEASE"DUMP" SWITCH
Test_Input_2(5)=0 * RELEASE"DUMP" SWITCH

**SCOPE PARAMETERS**
Task_List(1)=466
Task_List(2)=1219
Task_List(3)=1219
Task_List(4)=1219
Task_List(5)=1

**TARGET PARAMETERS**
Schedule
Initial_Proj_Deadline=94
Resistance_to_Sched_Slip=2
Quality
Project_Quality_Goal=1
Quality_Goal_Adjust_Time(1)=24
Quality_Goal_Adjust_Time(2)=24
Quality_Goal_Adjust_Time(3)=24
Quality_Goal_Adjust_Time(4)=24
Quality_Goal_Adjust_Time(5)=24

Cost
Avg_Straight_Pay(Phase)=25
Budget_Switch=1
Cost_Markup(Phase)=2
Overtime_Premium(Phase)=0.50
Percent_hourly_Labor(Phase)=0.00
Proj_Budget=500000

RESOURCE PARAMETERS
Gross Labor
HdctJumpStart(1)=1
HdctJumpStart(2)=1
HdctJumpStart(3)=1
HdctJumpStart(4)=1
HdctJumpStart(5)=1
HdctJumpStop(1)=1
HdctJumpStop(2)=1
HdctJumpStop(3)=1
HdctJumpStop(4)=1
HdctJumpStop(5)=1
HdctJumpSwitch(1)=0
HdctJumpSwitch(2)=1
HdctJumpSwitch(3)=0
HdctJumpSwitch(4)=0
HdctJumpSwitch(5)=0
Headcount_Adjustment_Time(1)=12
Headcount_Adjustment_Time(2)=8
Headcount_Adjustment_Time(3)=8
Headcount_Adjustment_Time(4)=8
Headcount_Adjustment_Time(5)=8
Initial_Headcount(1)=0.5
Initial_Headcount(2)=0.5
Initial_Headcount(3)=0.5
Initial_Headcount(4)=0.5
Initial_Headcount(5)=0.5
Initial_Quality_Goal(Phase)=0.9
Max_Headcount(1)=2
Max_Headcount(2)=2
Max_Headcount(3)=2
Max_Headcount(4)=2
Max_Headcount(5)=2
Max_Workweek(Phase)=140
Normal_Workweek(Phase)=40

Labor Allocation
BW_Labor_Delay(1)=1.5
BW_Labor_Delay(2)=1.5
BW_Labor_Delay(3)=1.5
BW_Labor_Delay(4)=1.5
BW_Labor_Delay(5)=1.5
BW_Prdctvty_Avg_Time(Phase)=1
BW_Prdctvty_Influences_Time(Phase)=1
BW_Priority(1)=3
BW_Priority(2)=3
BW_Priority(3)=5
BW_Priority(4)=3
BW_Priority(5)=3
Coord_Labor_Delay(1)=1
Coord_Labor_Delay(2)=1
Coord_Labor_Delay(3)=1
Coord_Labor_Delay(4)=1
Coord_Labor_Delay(5)=1
Coord_Priority(1)=1
Coord_Priority(2)=1
Coord_Priority(3)=1
Coord_Priority(4)=1
Coord_Priority(5)=1
QA_Labor_Delay(1)=12
QA_Labor_Delay(2)=3
QA_Labor_Delay(3)=0.5
QA_Labor_Delay(4)=3
QA_Labor_Delay(5)=7.75
RW_Labor_Delay(1)=1
RW_Labor_Delay(2)=1
RW_Labor_Delay(3)=1
RW_Labor_Delay(4)=1
RW_Labor_Delay(5)=1
RW_Priority(1)=1
RW_Priority(2)=2
RW_Priority(3)=1
RW_Priority(4)=1
RW_Priority(5)=1

Experience
Avg_New_member_Exper(Phase)=6
Exper_Assim_Time(Phase)=1

Productivity
Adjust_Expect_BW_Prdctvty_Time(Phase)=1
Avg_Act_BW_Prdctvty_Time(Phase)=1
Avg_Act_Coord_Prdctvty_Time(Phase)=1
Avg_Act_QA_Prdctvty_Time(Phase)=1
Avg_Act_RW_Prdctvty_Time(Phase)=1
Change_Expected_QA_Prdctvty_Time(Phase)=1
Ch_Expect_Coord_Prdctvty_time(Phase)=1
Coord_Prdctvty_Avg_Time(Phase)=1
Min_Exp_BW_Prdrag(Phase)=0.1
Min_Exp_Coord_Prdrag(Phase)=0.1
Min_Exp_QA_Prdrag(Phase)=0.1
Min_Exp_RW_Prdrag(Phase)=0.1
QA_Prdctvty_Avg_Time(Phase)=1
QA_Prdctvty_Report_Time(Phase)=1
QA_Priority(1)=1
QA_Priority(2)=2
QA_Priority(3)=1
QA_Priority(4)=1
QA_Priority(5)=1
Report_BW_Prdctvty_Time(Phase)=1
Report_Coord_Prdctvty_time(Phase)=1
RW_Prdctvty_Avg_Time(Phase)=1
Wt_to_Current_BW_Prdctvty(Phase)=1.00

MODEL CONTROL PARAMETERS
alpha(Phase)=100
CloseEnough 0.01
Complexity(Phase)=10
COORD_STATUS\(\text{test}(\text{Phase})=0.5\)
Deadline_Switch\(=1\)
LastPhase\(=3\)
Max_DL_Change(Phase)\(=100\)
Max_Proj_DL_Change\(=100\)
Min_Headcount(Phase)\(=0.001\)
PhaseNo\((1)=1\)
PhaseNo\((2)=2\)
PhaseNo\((3)=3\)
PhaseNo\((4)=4\)
PhaseNo\((5)=5\)
QA_STATUS(Phase)\(=1\)
Quality_Goal_Switch \(= 1\)
Ref_BW_Prdfctvty(Phase)\(=2\)
Ref_Complexity(Phase)\(=100\)
Ref_Coord_Prdfctvty(Phase)\(=1\)
Ref_Exper(Phase)\(=50\)
Ref_QA_Prdfctvty(Phase)\(=1.75\)
Ref_Qual_of_Practice(Phase)\(=5\)
Ref_RW_Prdfctvty(Phase)\(=1\)
ResourseSwitch(Phase)\(=1\)
TaskListScale(up,down)=Task_List(up)/Task_List(down)
Test_Input\(\_1(\text{Phase})=1\)  * ITERATION SWITCH
Test_Input\(\_3(\text{Phase})=1\)  * NOUSED
Time_to_Avg_Exp_Compl_Time(Phase)\(=1\)
Wrkwk_Avg_Time(Phase)\(=4\)