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[5.1].
- **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.

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:

[5.1] 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.

- 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 4:** Design - Translate product description into components and physical design using chip layout code. Review product specifications prepared in Phase 3: Product Definition and integrate product description and product design. Layout chip layers. Have tapes and masks for chip layers made.
- **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.

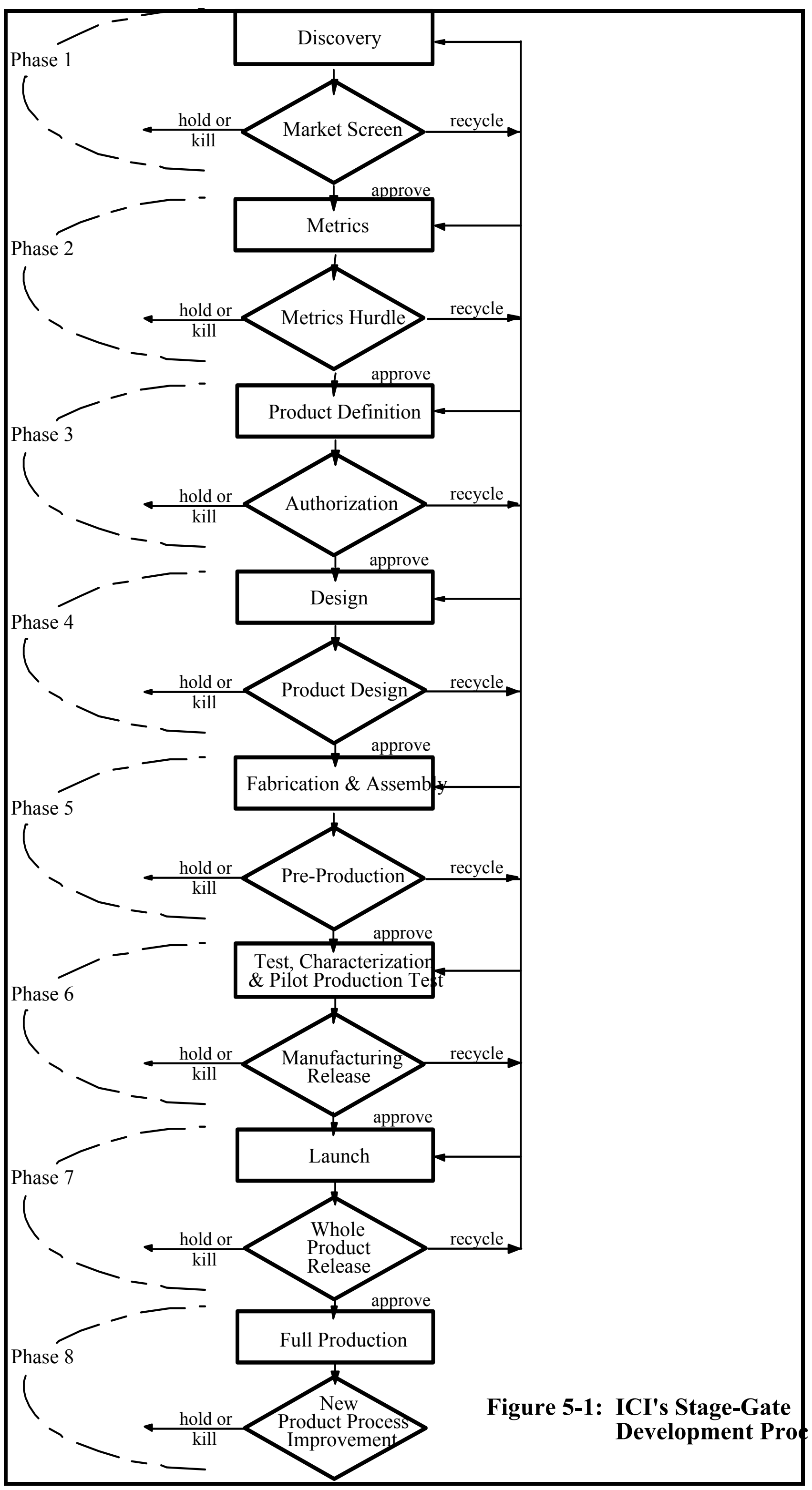

Figure 5-1: ICI's Stage-Gate Development Proc

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 Calantone, 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.

[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).

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."

5.3.6 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.

Turn figure found at end of this chapter sideways and attach here.

Table 5-1: Python Project Interfaces of Development Phases and Functional Groups

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.

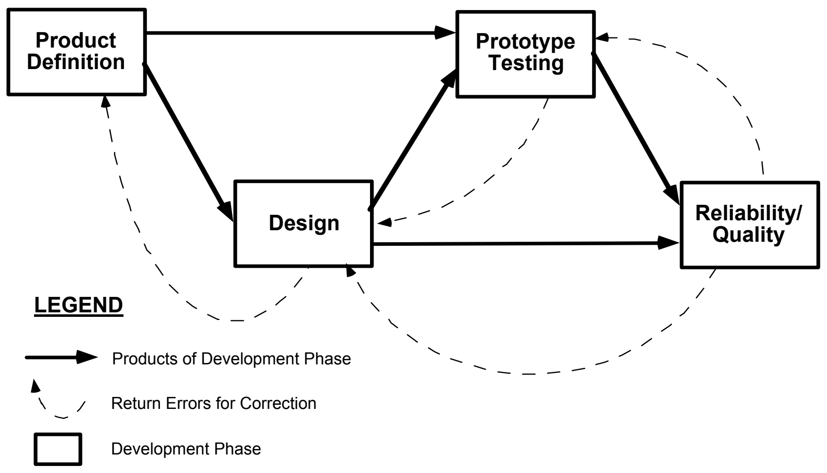

Figure 5-2: Project Phase Network for the Python Development Project

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[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.

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.

[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.

Project Variable	**Number of Times Identified**
The Development Process	
Information Flow	
Information sharing with market and customers	10
Quantity of information flow	8
Quality of information flow	6
Development Process Information Flow subtotal	24
Development Process Structure	
Stage-Gate structure (ICIDS)	7
Degree of concurrence in process	3
Development Process Structure subtotal	10
Changes	
General (role of, number of, etc.)	19
Quality Assurance	11
Rework	7
Start and Stop delays	1
Changes subtotal	38
Development Process Infrastructure & Support	
Tools available	6
Coordination of development process	5
Development Process Infrastructure & Support subtotal	11
Management of the Development Process	
Good product definition	9
Performance pressure	4
Systemic perspective of process by management	1
Management of Development Process subtotal	14
Development Process subtotal	**97**
Resources	
People	
Quantity available or used	25
Capabilities, knowledge, or experience	12
Delays in getting "up to speed" about the project	4
Project personnel turnover rate	1
Delays in getting the right people	1
People Resources subtotal	43

Table 5-2: Project Variable Types Generated by Interviews (partial)

Project Variable	**Number of Times Identified**
Non-human resources	
Quantity available or used	15
Capabilities	4
Delay due to unavailability	2

Non-human Resources subtotal	21
Other	
Unspecified resource type availability	8
Technology used in development	4
Resources: Other subtotal	12
Resources subtotal	**76**
The Development Organization	
Development Team	
Vision and direction of development team	9
Design of development team	7
Rewards for development team	7
Cohesion and morale of team	7
Team meetings	3
Systemic perspective of process by team	1
Development Team subtotal	34
Management and Leadership	
Support of team and project	6
Style and making of decisions	2
Delays in making decisions	1
Management and Leadership subtotal	9
Development Organization subtotal	**43**
The Development Environment	
The business cycle	1
The needs of the market	1
Development Environment subtotal	**2**
TOTAL VARIABLE IDENTIFICATIONS	**218**

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.

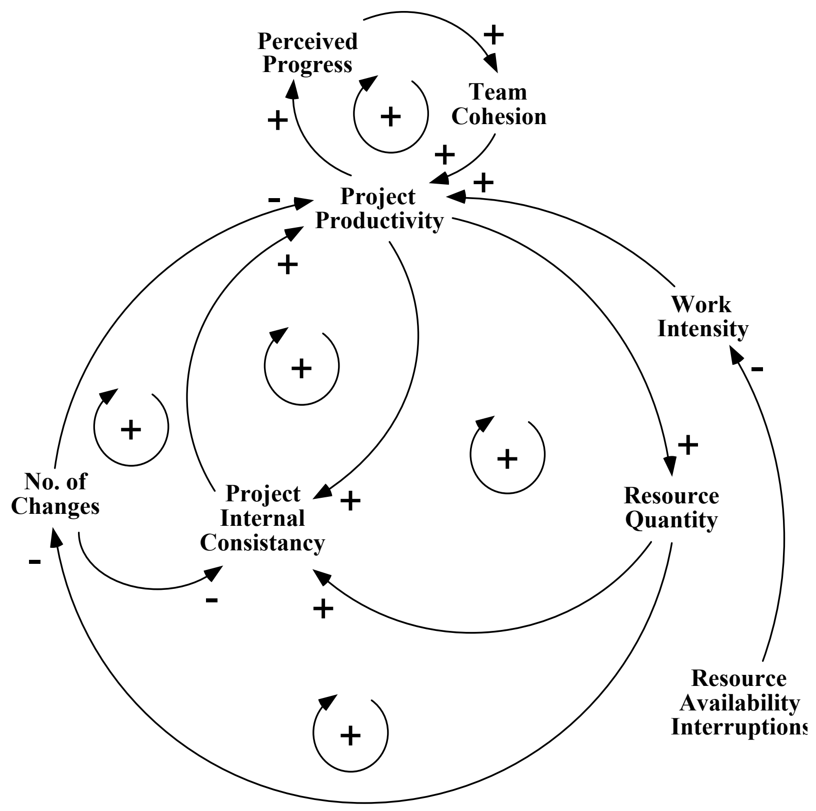

Figure 5-3: Primary Feedback Structures Identified at Model Boundary Workshop

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 unflawed. 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.

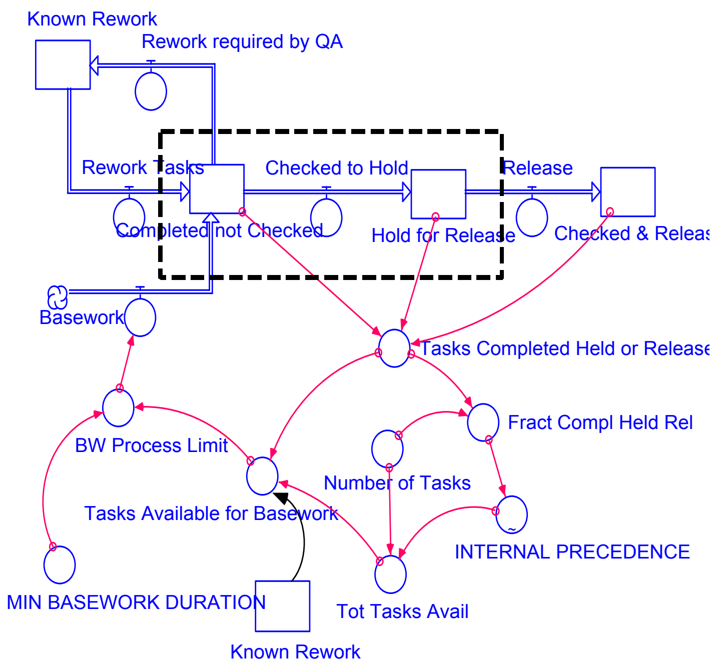

Figure 5-4: The Additional Stock Required to Model the Aggregate Release of Tasks from a Phase

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.

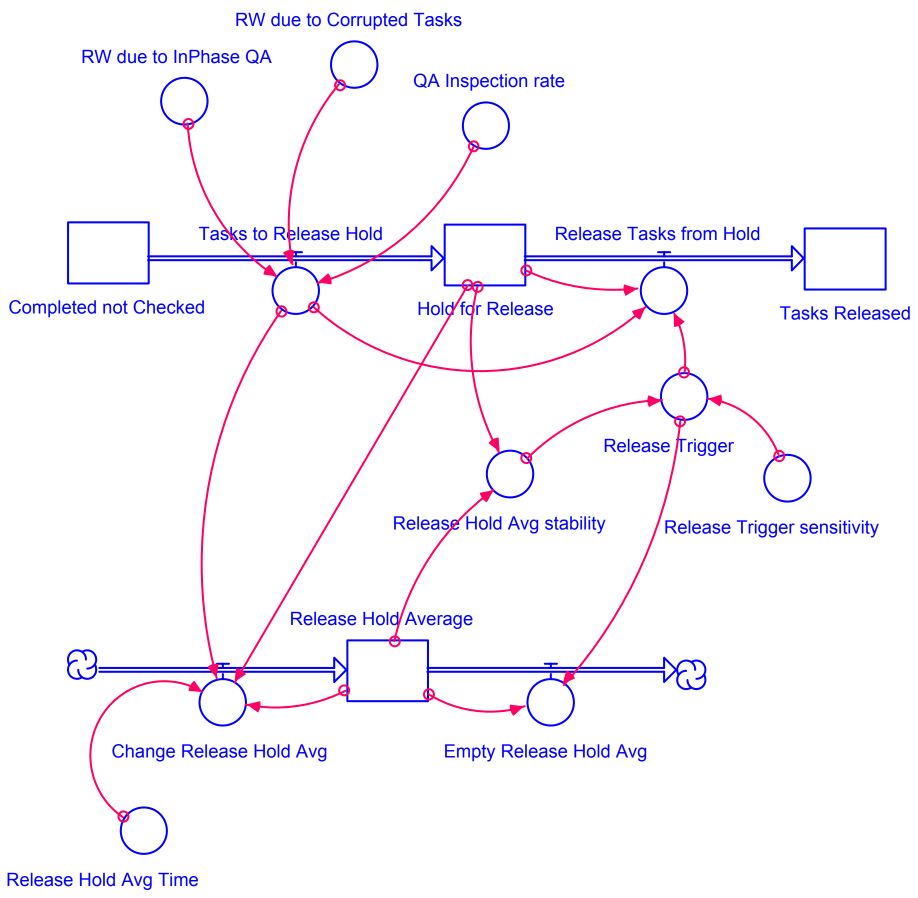

Figure 5-5: Additional Model Structure Representing the Aggregation and Holding of Development Tasks before Release

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.

Hold_for_Release(Phase)=Hold_for_Release(Phase)+dt*(Tasks_to_Release_Hold(Phase)-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.

Tasks_to_Release_Hold(Phase)=(QA_inspection_rate(Phase)-RW_due_to_InPhase_QA(Phase)-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).

```
Release_Tasks_from_Hold(Phase)=Release_Trigger(Phase)*((Hold_for_Release(Phase)/dt)+Tasks_t
o_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.

```
Release_Trigger(Phase)=FIFGE(0,1,MAX(Release_Hold_Avg_Stability(Phase),
(-1)*Release_Hold_Avg_Stability(Phase)),Release_Trigger_Sensativity(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.

```
Release_Hold_Avg_Stability(Phase)=Release_Hold_Avg(Phase)-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.

```
Release_Hold_Avg(Phase)=Release_Hold_Avg(Phase)+
dt*(Change_in_Release_Hold_Avg(Phase)-Empty_Release_Hold_Avg(Phase))

Change_in_Release_Hold_Avg(Phase)=((Hold_for_Release(Phase)+
Tasks_to_Release_Hold(Phase)-Release_Hold_Avg(Phase))/
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.

```
Empty_Release_Hold_Avg(Phase)=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[5.4] 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

[5.4] The other type of parameter is model control parameters. See Appendix 5.1 for a complete listing of parameter values for calibration.

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.

<u>Process Parameters</u>
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.

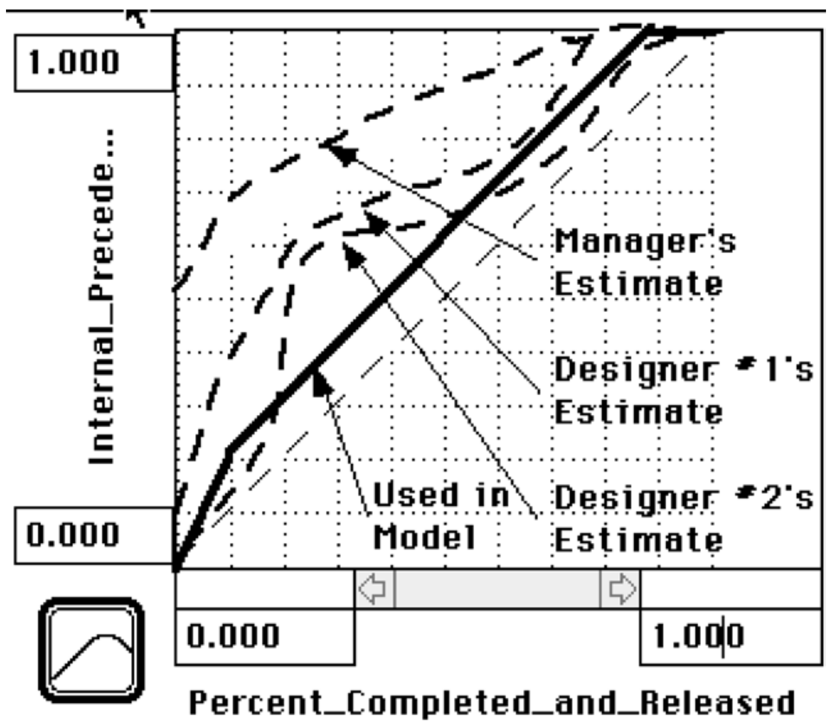

Figure 5-6: Design Phase Internal Precedence Relationship for Calibration

Release_Trigger_Sensativity(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_Proj_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

Inital_Headcout(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 manger: "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.

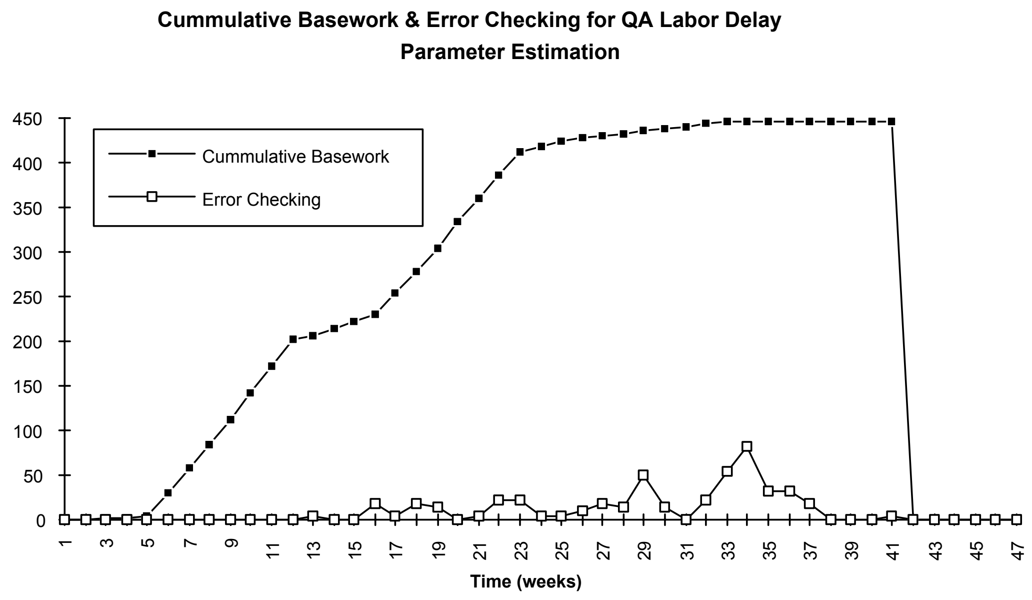

Figure 5-7: Cumulative Basework and Error Checking for QA labor Delay Parameter Estimate

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.

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

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.

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).

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) =
0.4/0.5/0.6/0.7/0.8/0.9/0.95/1.0/1.0/1.0/1.0
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) =
1.0/1.0/1.0/1.0/1.0/1.0/1.0/1.0/1.0/1.0/1.0

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/.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) =
0.0/0.0/0.0/0.0/0.0/0.0/0.0/0.0/0.0/1.0/1.0
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
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.

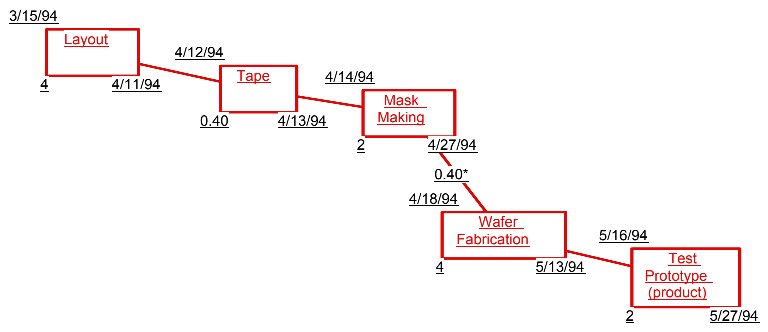

Figure 5-8: Critical Path Method Estimate of Static Delay between Design and Prototype Test Phases: Activity Chart

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.

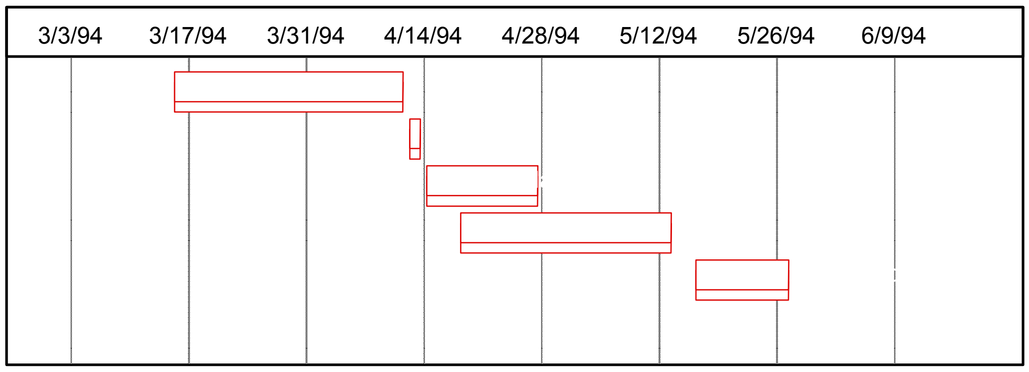

Figure 5-9: Critical Path Method Estimate of Static Delay between Design and Prototype Test Phases: Activity Timeline

As shown in the Activity Timeline, the static delay is typically a 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) =
0.0/0.0/0.0/0.0/0.0/0.0/0.0/0.0/0.0/1.0/1.0
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) =
0.0/0.0/0.0/0.0/0.0/0.0/0.0/0.0/0.0/1.0/1.0
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.

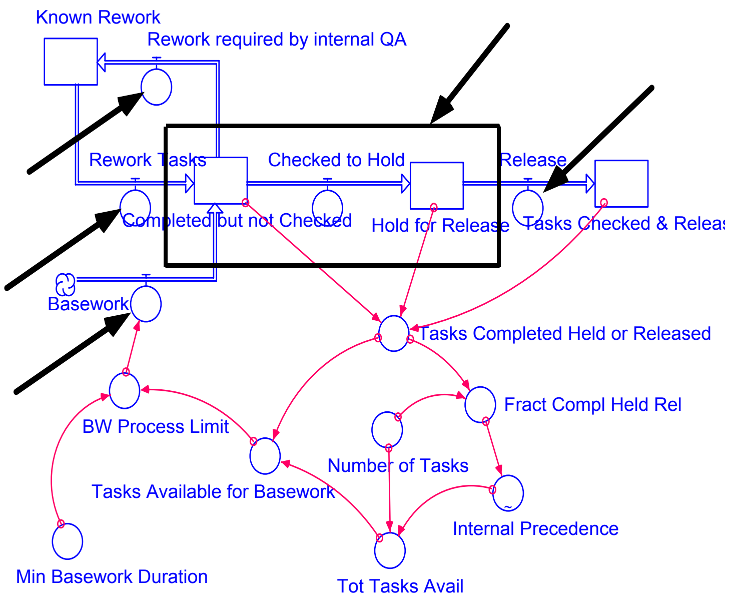

Figure 5-10: Design Phase Model Calibration to Historical Behavior

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.

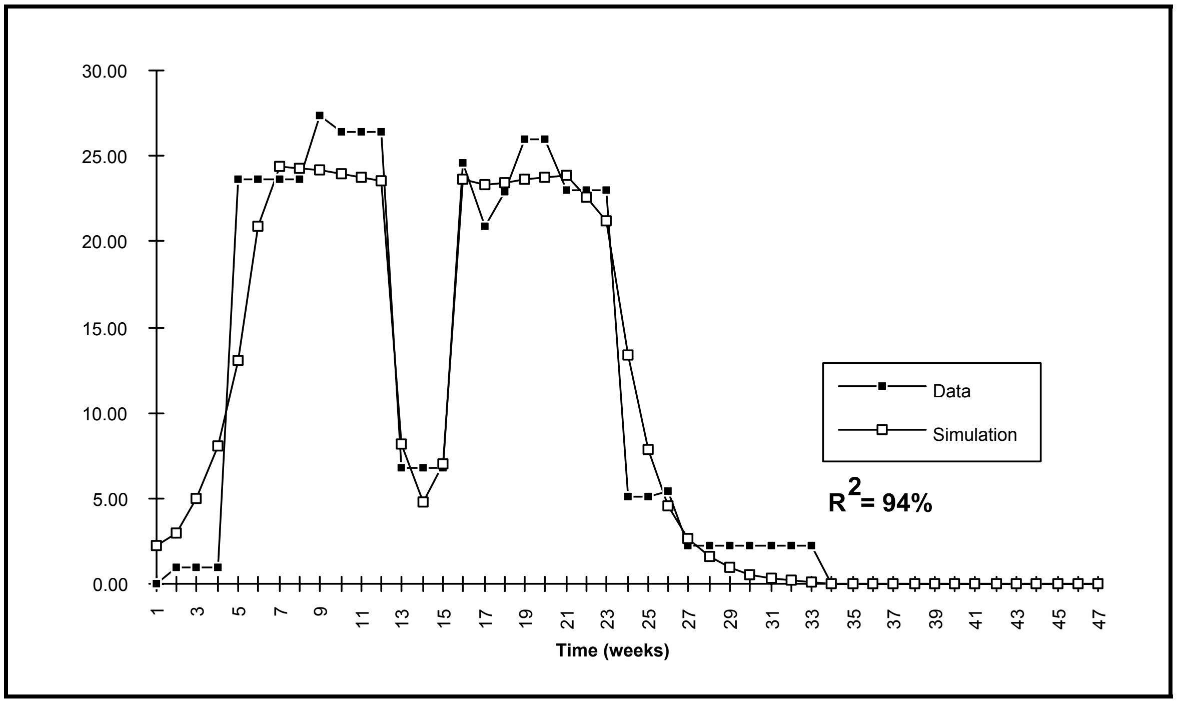

Figure 5-11: Design Phase Model Calibration to Historical Behavior Basework Rate

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.

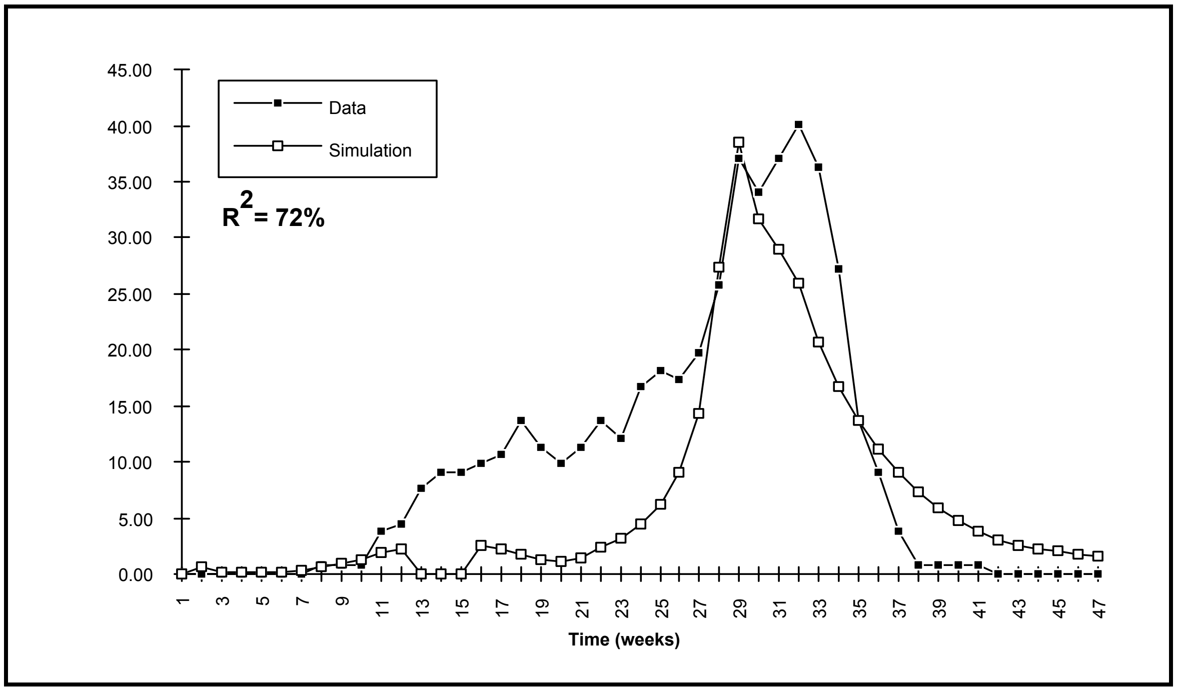

Figure 5-12: Design Phase Model Calibration to Historical Behavior Rework due to Internal Quality Assurance Rate

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 Basework 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 basework 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.

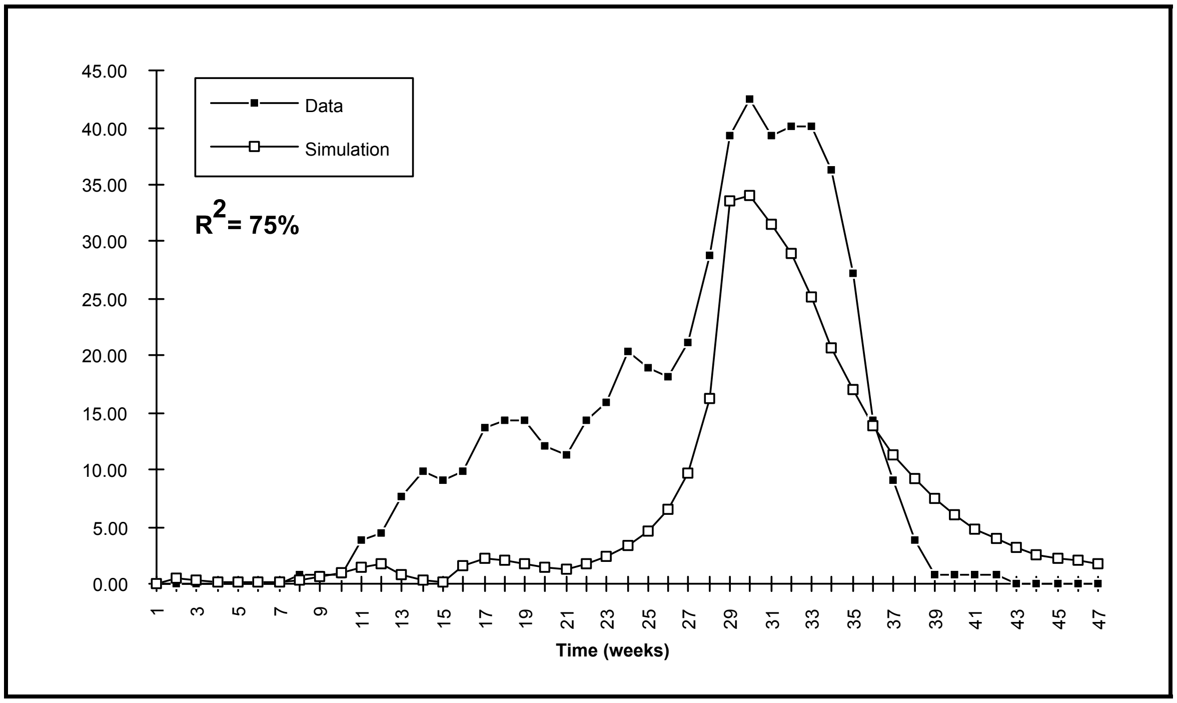

Figure 5-13: Design Phase Model Calibration to Historical Behavior Rework Rate

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.

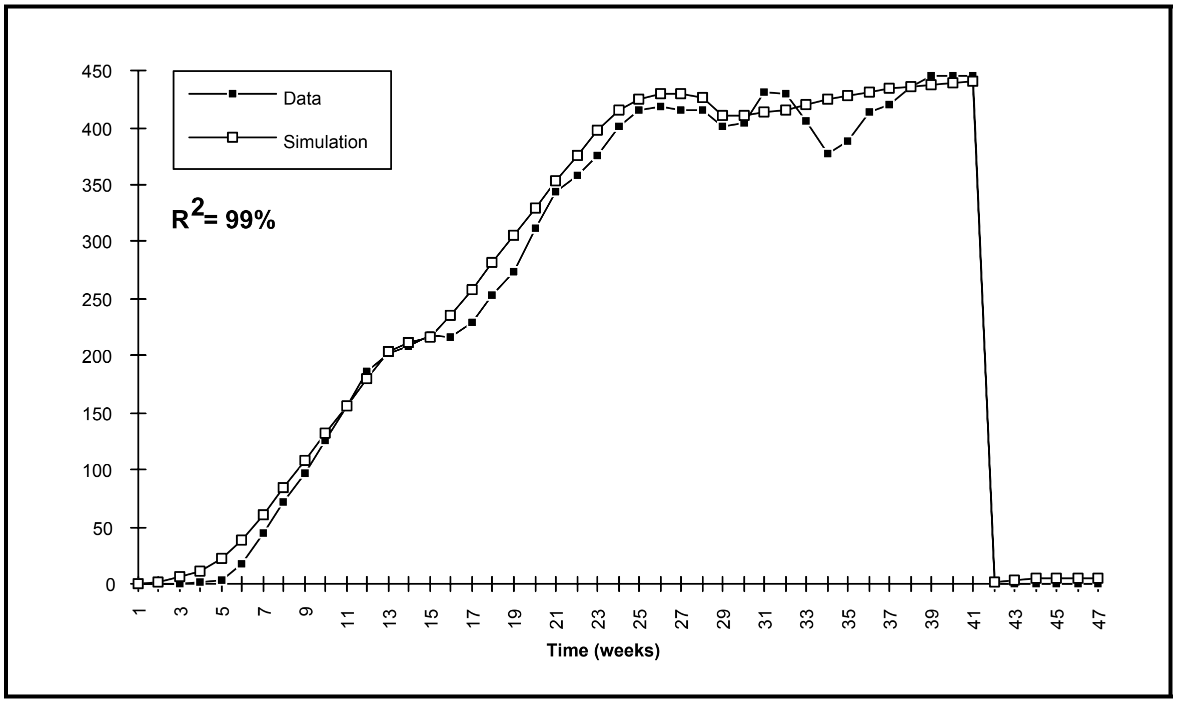

Figure 5-14: Design Phase Model Calibration to Historical Behavior Tasks Completed but not Checked and Tasks Held for Release

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.

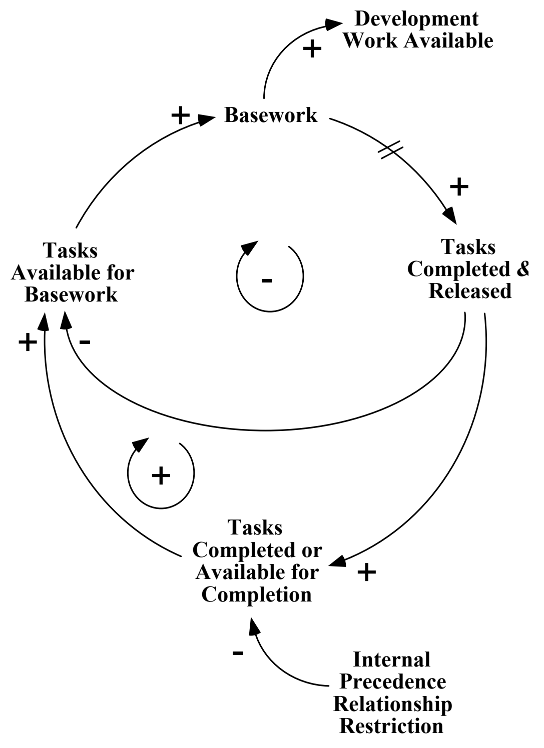
Development Work Available
+
Basework
+
+
Tasks Available for Basework
-
Tasks Completed & Released
+
-
+
Tasks Completed or Available for Completion
+
-
Internal Precedence Relationship Restriction

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.

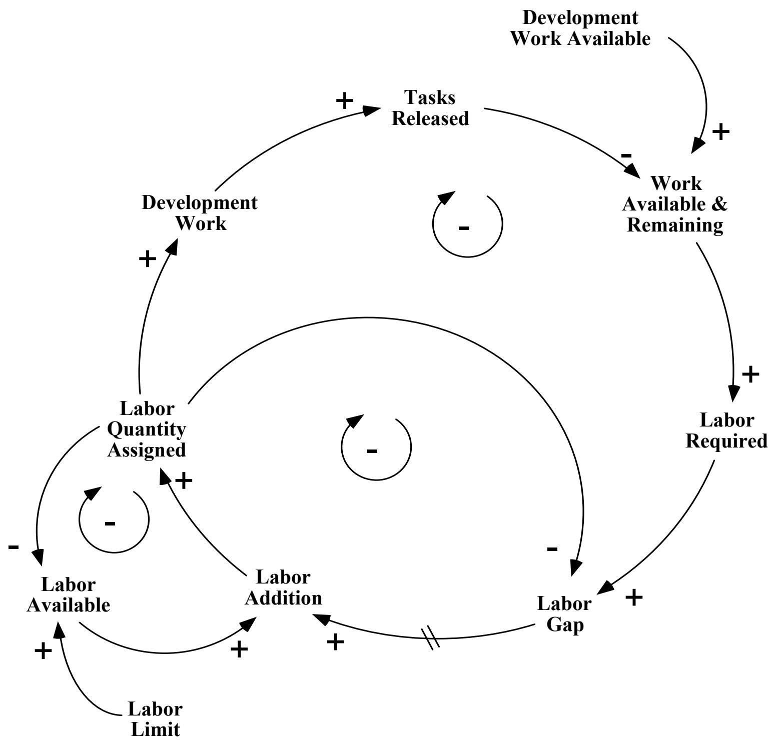

Figure 5-16: Single Phase Labor Availability Feedback Structure

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.

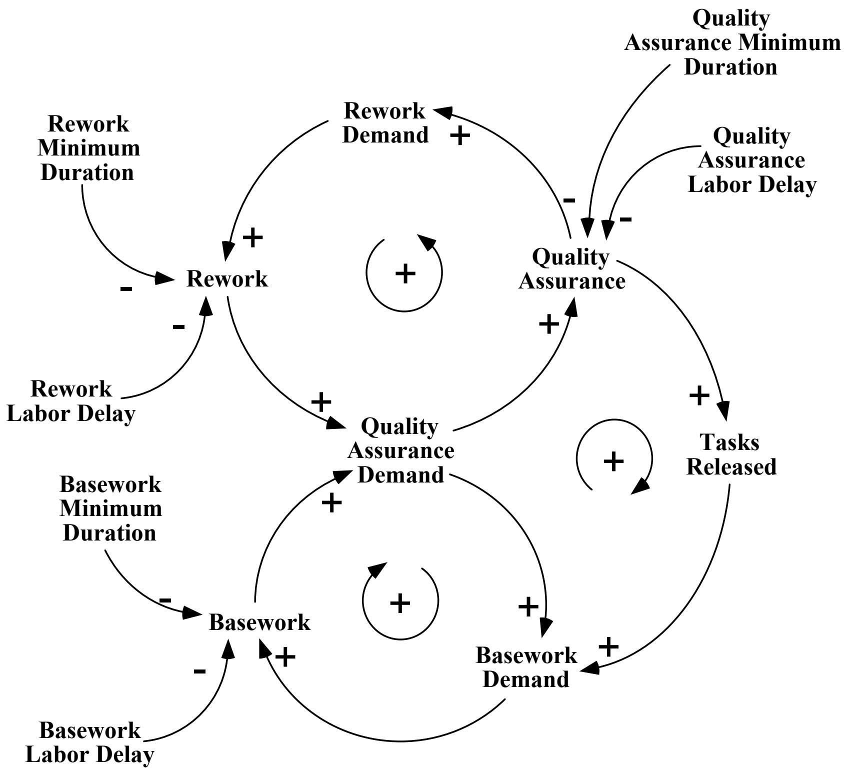

Figure 5-17: Single Phase Error Feedback Structure

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.

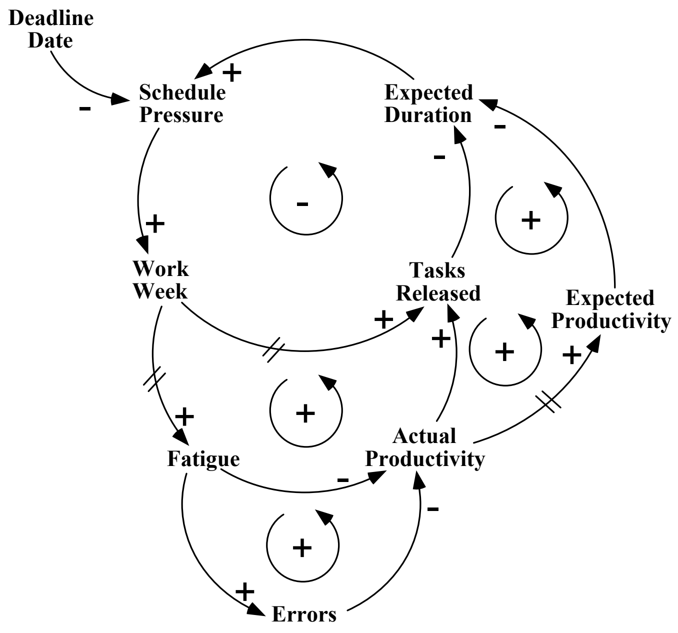

Figure 5-18: Compensating Feedback Loops in Schedule Project Target Structure

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.

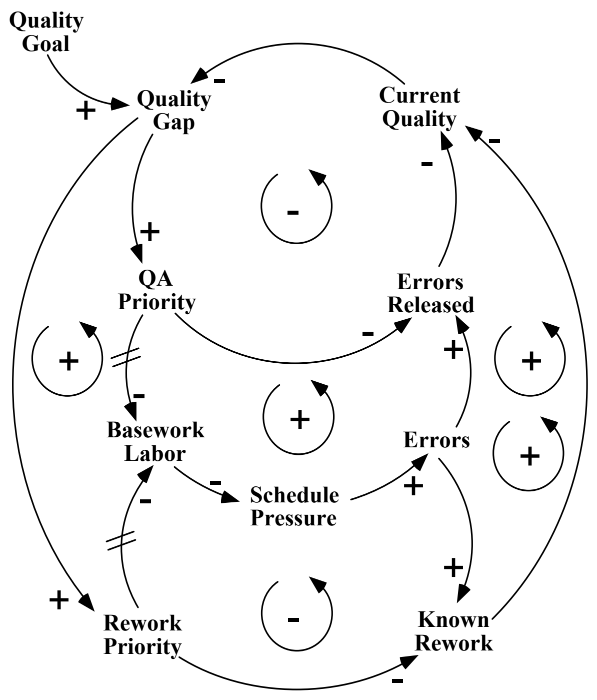

Figure 5-19: Compensating Feedback Loops in Quality Project Target Structure

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.

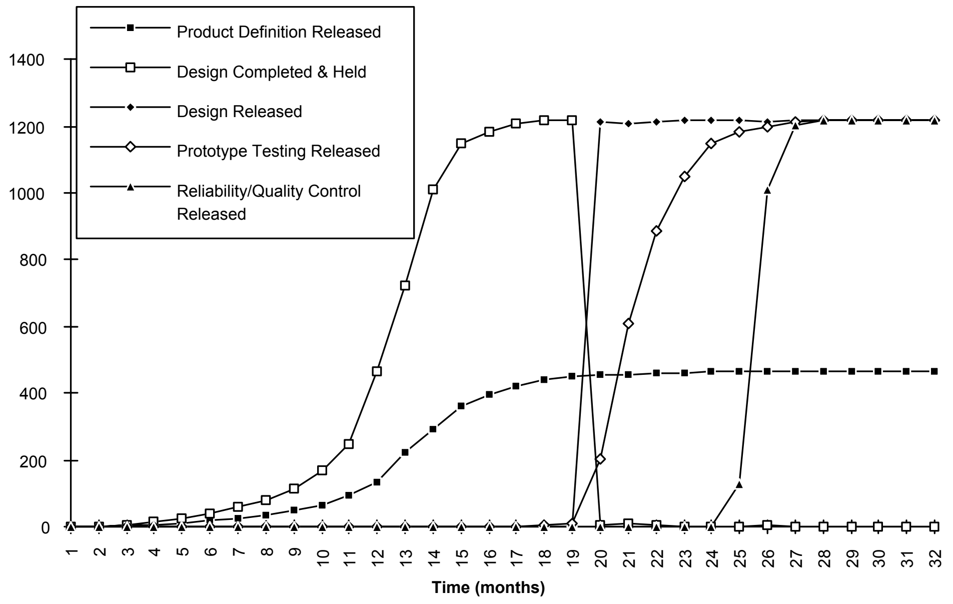

Figure 5-20: Multiple Phase Project Simulation

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.

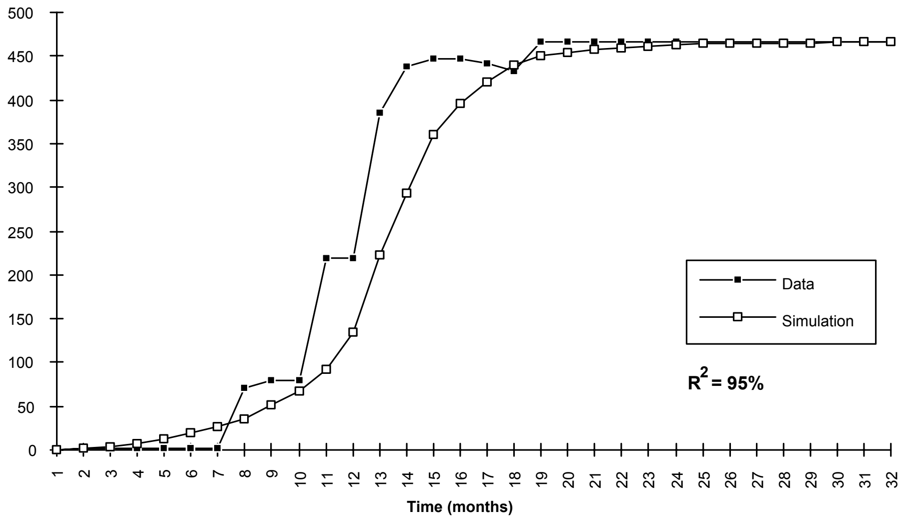

Figure 5-21: Multiple Phase Model Calibration to Historical Behavior Product Definition Tasks Released

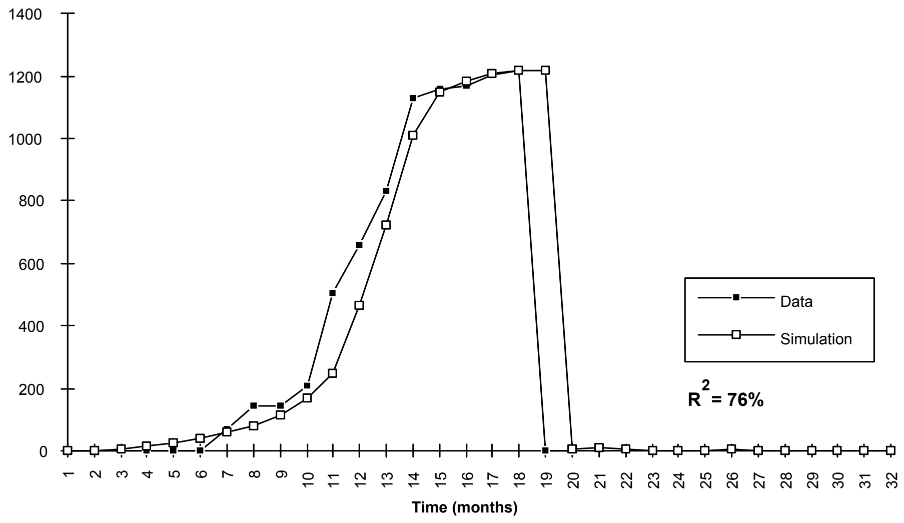

Figure 5-22: Multiple Phase Model Calibration to Historical Behavior Design Tasks Completed and held for Release

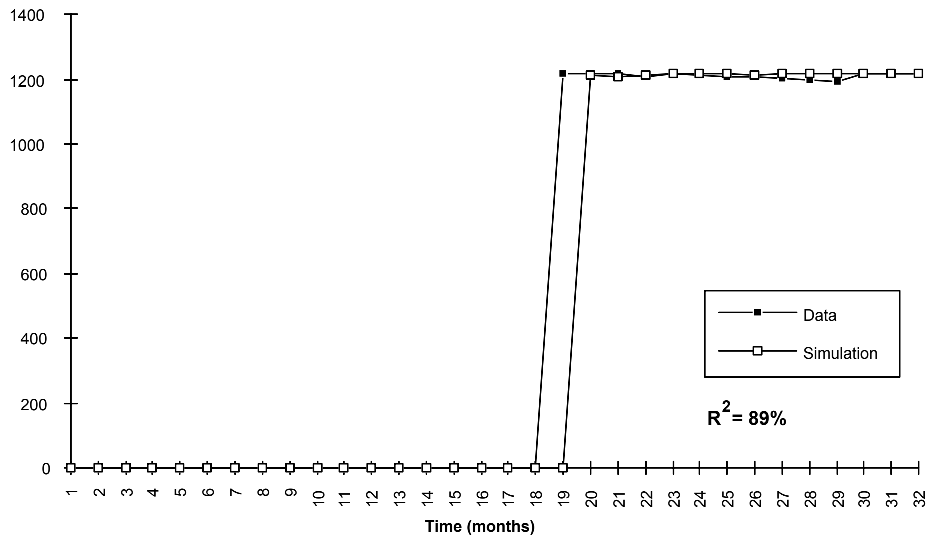

Figure 5-23: Multiple Phase Model Calibration to Historical Behavior Design Tasks Released

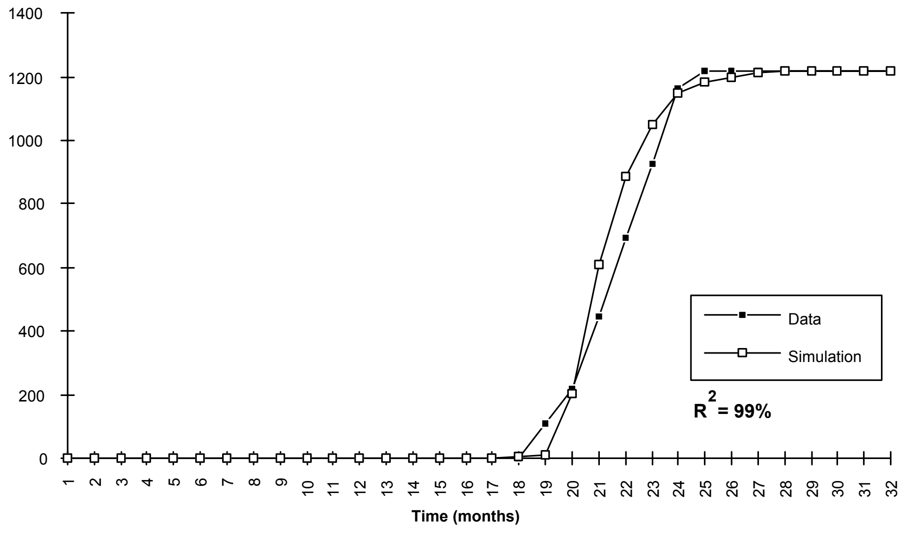

Figure 5-24: Multiple Phase Model Calibration to Historical Behavior Prototype Testing Tasks Released

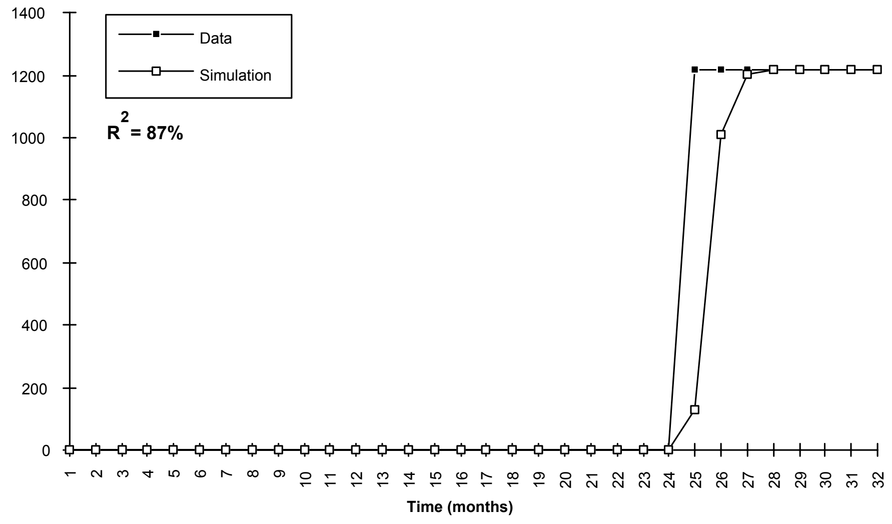

Figure 5-25: Multiple Phase Model Calibration to Historical Behavior Reliability/quality Control Tasks Released

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.

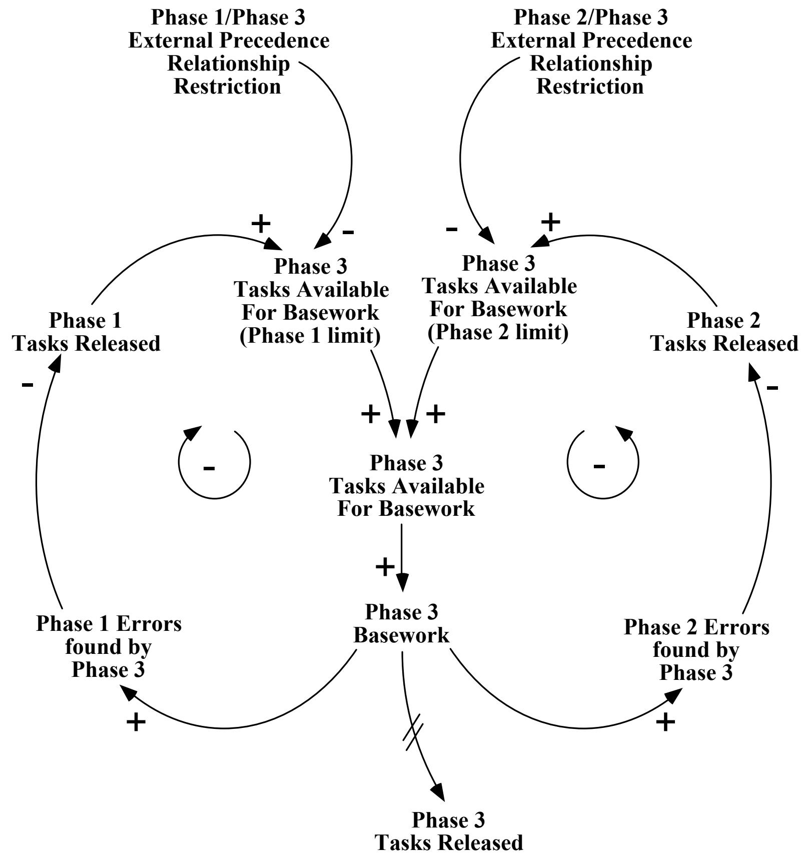

Figure 5-26: Inter-Phase Available Work Feedback

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 basework in Phase 3. The result is a reduction in Phase 3 basework 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.

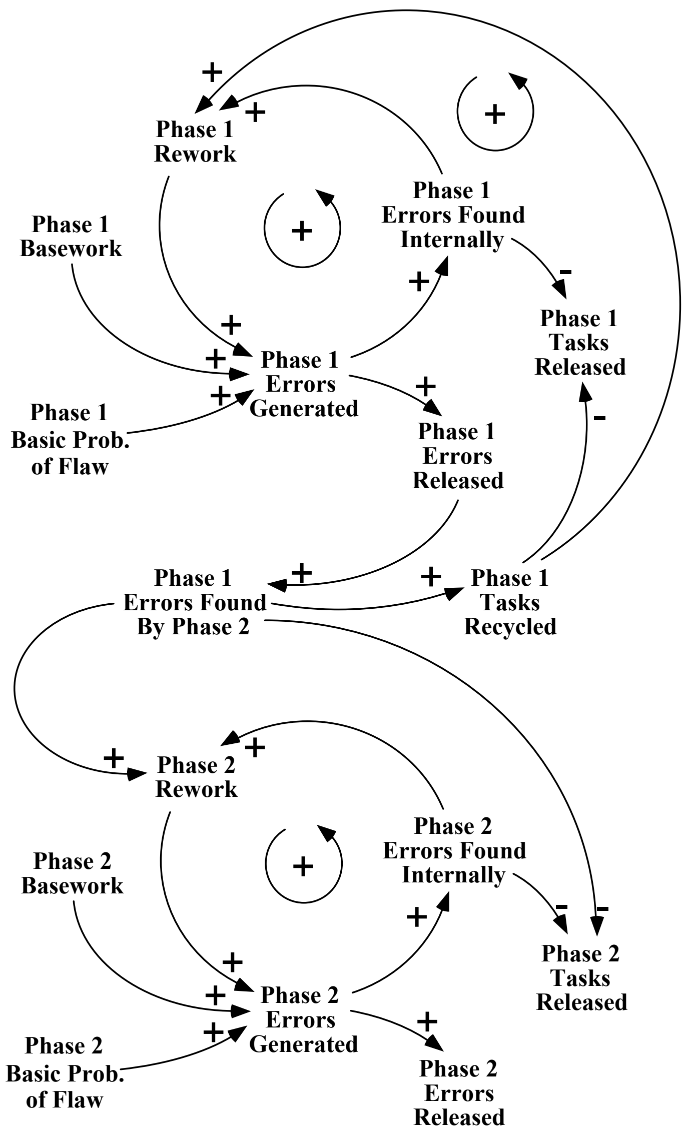

Figure 5-27: Multiple Phase Model Error Flows

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.