Lean Manufacturing Techniques Applied to Software Development

by

Thane Morgan

BSEE, 1984 University of Colorado

SUBMITTED TO THE SYSTEM DESIGN AND MANAGEMENT PROGRAM IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR THE DEGREE OF

MASTER OF SCIENCE IN ENGINEERING AND MANAGEMENT

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

January 1998 [June 1998]

Signature of Author:_____
System Design and Management Program
January 16, 1998

ARCHIVES

Certified by:_____
Nelson P. Repenning
Robert N. Noyce Career Development Assistant Professor
Operations Management/System Dynamics Group
Thesis Supervisor

Accepted by:_____
Thomas A. Kochan
Acting Codirector, System Design and Management Program

Accepted by:_____
John R. Williams
Codirector, System Design and Management Program

Lean Manufacturing Techniques Applied to Software Development

by

Thane Morgan

Submitted to the System Design and Management Program on January 16, 1998 in partial fulfillment of the requirements for the Degree of

Master of Science in Engineering and Management

ABSTRACT

Operation of a lean manufacturing assembly cell is studied and modeled using system dynamics. A case study of lean software development is provided and used as a guide in building a system dynamics model integrating structures from the assembly cell model and a standard project model. Both models are analyzed, and the applicability of lean manufacturing ideas and techniques to software development is demonstrated.

Thesis Supervisor: Nelson P. Repenning
Title: Robert N. Noyce Career Development Assistant Professor
Operations Management/System Dynamics Group

Table of Contents

I. Introduction:

This investigation focuses on the applicability of lean manufacturing techniques to software development. It is motivated by my years in software development for real-time embedded applications and my studies at MIT. Over the years I, along with many colleagues, have developed a set of ideas and practices for running software development projects that are very successful. We have achieved very high levels of software reuse, productivity, quality, and schedule predictability.

At MIT, I had the pleasure of taking several courses from the department of mechanical engineering. With the guidance of Prof. David Cochran, and his course in manufacturing system design, I was introduced to the principles of the Toyota Production System (TPS) [10] [12] [13] [16]. In the process of learning TPS, I found that there was a great deal of similarity between it and the ideas and practices we had developed for guiding software development. The ideas of TPS are much more mature, but I could see that we had begun to stumble down a similar path.

At the same time I was studying lean production with Prof. Cochran, I was being introduced to the field of system dynamics [4] [5] [15] by Prof. Nelson Repenning. I found it an exceptionally useful tool in understanding human systems. It occurred to me that I might be able to use SD modeling to explore my perceived similarities between manufacturing and software development. While my studies are focused primarily on software development, I believe that the ideas are applicable to product development in general.

I attended a lecture by James Womack, co-author of "Lean Thinking," that fueled my convictions. In his book [14] he generalizes the principles of lean manufacturing and goes beyond the more prescriptive presentations supplied by Ohno, Monden, and Shingo. Although he continues to focus on manufacturing, by separating principles and implementation he provides a higher level of abstraction that is easier to apply in other disciplines.

I have approached the topic in the following manner. First I have developed an SD model of a lean manufacturing system. The model is designed to capture the basic concepts of lean manufacturing as outlined in the literature [1] [10-13] [16] [17] and reproduce the widely accepted modes of behavior. Second, I have written a case study outlining the ideas, processes, and results of one of my most recent software development projects. Finally, I have combined a basic project model [2] [8] with the basic loops from the manufacturing model as justified by my case study.

The final analysis of the combined model provides some compelling evidence that the ideas of lean manufacturing are indeed applicable, in principle, to software development. The data will show that lean software development exits and can be practiced.

II. Lean Manufacturing:

A. General Background:

Lean manufacturing is a term commonly used to describe the manufacturing ideas and processes developed by Toyota Motor Company. Taiichi Ohno and others developed what he called the Toyota Production System (TPS) starting in 1945. Many excellent books have been written on the subject [10-13] [16] [17]. In these pages I will attempt to describe the high-level concepts and processes relevant to this investigation.

Lean manufacturing is based on two fundamental principles: customer value and the elimination of waste. These principles give rise to a variety of methods of implementation. The most important ones for this study are kanban, jidoka, process improvement, and standard work.

B. Customer Value

Customer value is the starting point for lean manufacturing. Since the point of manufacturing is to provide a product in exchange for the customer's money, and value is the only thing that customers want for their money, the only thing worth producing is what customers perceive as value. The idea of customer value has two components: delivering value and the profit equation.

The profit equation is the following:

$$\text{Profit} = \text{Price} - \text{Cost}.$$

While seemingly obvious, Taiichi Ohno makes an important point in that how the equation is solved is very important. A common way to solve the equation is to decide on a desired profit, measure cost, and calculate the price. The lean manufacturing approach is to accept customer value as the price and profit as the difference between price and cost. This leaves only cost under the control of the producer. Since cost is the only control point, all efforts must be focused on reducing it and maximizing profit. (An added desirable effect is decoupling the pricing policy from the product cost [14].)

The delivering value comes from the idea of customer-set pricing and cost as the manufacturer's control point. Ohno defines the concept as the first step in deriving, TPS. Womack and Jones define it as the guiding principles for lean thinking. Using slightly different terminology, they break it down into five steps.

1. Determine customer value -- This step is essentially product selection and good marketing. If the customer wants a light blue, left-handed, voice-activated letter opener, then don't make it red. (Easily said, difficult to implement.)
2. Identify the value stream -- This means understanding the steps/process/materials required to produce the previously determined customer value.
3. Make the value stream flow -- This step really only makes sense in the context of the next one. When a customer pulls their desired value from the stream, it

is replaced. The best analogy for this is a river, where water flowing downstream is immediately replaced by more water from upstream.

4. Deliver exactly, and only, what the customer wants in the desired quantity at the desired time -- This is meeting customer needs precisely, which represents the breakpoint in the profit equation. Since price only increases up to the point at which the customer has received the desired value, the added cost of providing excess value only serves to reduce profit.
5. Pursue perfection -- Never accept the current state of the system as good enough. While the system must be operated in its current state, never stop working on ways to make it better.

C. Elimination of Waste

A direct response to the principles of customer value is the elimination of waste. Waste is defined as anything that does not add value. Ohno defines seven types of waste in manufacturing:

1. Overproduction -- This is the classic problem of making 100 widgets when only 50 can be sold. The cost of producing the 51st and each successive widget only serves to reduce profit.
2. Time on hand (waiting) -- Waiting, by definition, produces no value, hence all waiting is waste and should be eliminated.
3. Transportation -- Moving parts and people adds no value to the product.
4. Processing itself -- There are usually many different ways to produce the desired value. The process currently being used is wasteful if there is another, less costly one that can produce the same value.
5. Stock on hand (inventory) -- While sometimes useful for buffing purposes, inventory is a significant investment that consumes resources that could be used elsewhere. Any inventory in excess of the minimum required is waste.
6. Movement -- Movement during assembly or processing is similar to transportation but focuses on worker motion. Excess or unnecessary motion is waste.
7. Making defective products -- Producing scrap consumes the same resources as producing actual products. Opportunity costs are involved, since a good product could have been produced as well with the same labor, machine wear, and energy.

Identifying and eliminating these wastes is the focus of all improvement efforts in a lean manufacturing system.

D. Kanban

Kanban is a simple but powerful concept for controlling a manufacturing process. There are many variations on the theme, but the basic kanban is a card associated with the product to be

produced. A completed product has its card attached to it. When the product is removed for use, the kanban card is removed and used to signal the need to replace the just-consumed part. The number of kanban cards determines the amount of inventory. Without an authorizing kanban card, no work is performed.

The function of the kanban system is summarized by Ohno in Table 1:

Function	Rules for Use
Provides pickup or transport information.	Later process picks up the number of items indicated by the kanban at the earlier process.
Provides production information.	Earlier process produces items in the quantity and sequence indicated by the kanban.
Prevents overproduction and excessive transport.	No items are made or transported without a kanban.
Serves as a work order attached to goods.	Always attach a kanban to the goods.
Prevents defective products by identifying the process making the defectives.	Defective products are not sent on to the subsequent process. The result is 100% defect-free goods.
Reveals existing problems and maintains inventory control.	Reducing the number of kanban increases their sensitivity.

Table 1, Kanban Functionality

Kanban are designed to provide the information needed to drive jidoka and process improvement. High levels of inventory mask that information, so one of the hallmarks of lean manufacturing is lowering the levels of inventory.

E. Jidoka (Mistake-proofing)

Jidoka is the process of putting in place automatic prevention of mistakes. It is the manufacturing embodiment of Franklin's truism, "An ounce of prevention is worth a pound of cure." The original idea came from special weaving machines that were designed to shut down when a thread broke. By shutting down, the machines told the operator to fix the problem and prevented the production of useless cloth. With proper design, a large number of the possible problems associated with a process can be eliminated.

F. Process Improvement

Process improvement is similar to jidoka except it is not automatic. It involves the development of new processes (that may be subsequently mistake-proofed), refinement of procedures for current processes that cannot be automated, substitution of a better material, or anything else that reduces waste.

G. Standard Work

Standard work is just what it says. Every task or operation is clearly defined and documented in a visible location where it is being performed. There are several important aspects of standard work. The most important is that it is defined by the people who perform it. Managers at a desk do not possess the knowledge or understanding to define the work. Only the people who actually do the work possess the required knowledge.

The beneficial aspects of standard work are establishing a baseline for future improvement and ease of worker transition. Random processes produce random results. With a baseline work definition, the results are consistent, repeatable, and thus identifiable. Worker transition is desirable, since it increases work-force flexibility.

III. Case Study:

A. General Background:

Cummins Engine Company was founded by the driver of the local banker in Columbus, IN. Clessey Cummins started experimenting with diesel engines around 1910. Cummins became firmly established as a major supplier of engines after WW II. From 1910 until the late 1980s, diesel engines were basically hydro-mechanical devices. With the spread of emission control regulations from passenger to industrial (trucking, mining, construction, etc.), electronic control of fueling and timing became common.

Electronics changed the nature of the diesel engine business. The features contained in the software became the major source of product differentiation. The ability to translate customer requests into value-added features became vital in order to maintain sales. The technology of designing and developing the engine controller software had to be mastered in order to simply maintain market share. Electronics-related features now consume over one third of Cummins's research and development budget.

Once a computer was attached to the engine, functions other than basic fueling and timing became possible. Safety features like limiting vehicle speed, diagnostic features like self-testing and monitoring, and high-tech features like variable horsepower ratings appeared. The added functionality rapidly went from "nice-to-have" to required status. From there the feature race was on. The three major players in the industry (Cummins, Caterpillar, and Detroit Diesel) continue to battle to see who can design and market the next innovative feature.

In the early eighties, with roughly 60% of the heavy duty trucking market, Cummins was the dominant manufacturer of diesel engines prior to the introduction of electronic controls. By early 1992, Cummins was still number one, but with 35% of the market. Mechanically faulty products introduced in 1988, along with a late and troubled release of electronically controlled products, had taken their toll.

Cummins was successfully selling its first-generation electronically controlled diesel engines. The system, known internally as Celect 91 and externally as CELECT, was working well, and customers were generally pleased. Cummins Electronics, a wholly owned subsidiary of Cummins Engine Company, was working on the second generation, known internally as Celect 94.

From a functional viewpoint, Celect 94 was supposed to be an upgraded version of Celect 91. In implementation, Celect 94 was completely new. It had been started in 1992 in response to exploding customer needs. The hardware had more memory and a faster processor. The software was to be written in the 'C' programming language instead of assembly to speed development and reduce maintenance costs.

In January 1993, Celect 94 hit a wall. By that time it was obvious that, as designed, it wasn't even capable of replicating Celect 91, let alone surpassing it. Cummins found itself in the position of being unable to recreate a product it had spent millions developing scant years earlier. Celect 94 could be delayed, but it could not be allowed to fail.

In an attempt to save the program, Celect 94 was reorganized. New leaders were put in place, selected where possible from the best that Cummins had available. Where no track record was available, people who were willing and had a plan were given an opportunity to prove themselves. The entire team was moved to a separate building. No justifiable request for resources were denied. Fingers were crossed and performance closely monitored.

The reorganized program had many exciting times, but eventually, was very successful. So successful, in fact, that it was selected as a platform for developing future electronic products. One of the main reasons cited for the decision was the strength of the processes created by the reorganized project team.

B. Celect 91 Development

While some characterized the software development process at Cummins as non-existent, it was a typical first-generation process. The development team would define the required functions, code some, test some, fix some bugs, modify the functionality, etc. After enough iteration, the product was considered adequate and put into production.

Two problems with the technique were discovered after the fact. First, no permanent record of requirements existed except in the code. When Celect 94 was started and the new software people asked for the requirements, the answer came back, "Make it like '91." Although succinct, the statement was not quite accurate. What was really meant was, "Make it like '91 except for a few improvements that might or might not conflict with existing functionality."

Second, the process took too long. The original product took eight years to develop. The technical challenge of building a new fuel system effectively hid the inefficiencies of the software development process. Thus when Celect 94 was launched with the same processes and the added burden of meeting or exceeding Celect 91, the stage was set for disaster.

C. Philosophy for New Software Processes

The details of the processes that were developed on the Celect 95 program are important. Later in this case they will be explained completely. However, the details are developed from a set of principles that set the tone and define a philosophical context. Everything that was created in the way of a process was guided by these ideas. They are extremely important because the details will change from one situation to another, or even over time on the same project, but the principles are constant.

1. Focus on result

Guiding all of the work was a conviction that what counted was the end result. The end user of an engine does not care about the controller software. They are interested in engine performance. Software and software engineering are only a means of delivering that performance. Too often, development teams get caught up in the elegance of the software. It is a complete waste of time. Every task, activity, and decision must be evaluated to make sure it contributes to delivering the end product or being able to continue to deliver the product. Those processes that do not add value should be eliminated.

2. Normalize Data:

This principle is based on two observations. First, two copies of the same information rarely agree, and if they do it will only last for a short period of time. Second, duplicate information that does not agree is a chronic and serious source of error. The answer is to keep only one copy of any piece of data. The practice means that a tree of usage must be created, so that when the data is changed it will ripple automatically through the system.

One example from our system is the CPU clock speed. It is used throughout the initialization process. There is only one file where the definition is kept, and every aspect of the system that depends on the clock speed references that file and uses the value in determining dependent values. The resulting hierarchy of information makes the task of upgrading the processor speed simple and quick.

3. Provide Feedback, Quickly:

The output of any activity has two facets: desired results and actual results. "Desired" and "actual" are usually close but rarely equal. Quality represents the difference and can only be improved if it is measured. Providing the feedback quickly is important because it prevents further defect creation in similar and dependent work. Good work based on a faulty base usually has to be thrown out, as well.

4. Iterate:

Plan for rework. Quality is never perfect and perfection is rarely cost-effective. As shown in the table, three passes at 70% are better than one pass at 95%. While not a replacement for high quality, iteration raises the effective quality from the current level. It also helps with the law of diminishing return. At some point the marginal cost of increasing first-time quality will exceed the benefit. Iteration can achieve the higher quality at a lower (usually) cost.

Completion Percentage	First Iteration	Second Iteration	Third Iteration
50%	50%	75%	88%
60%	60%	84%	94%
70%	70%	91%	97%
80%	80%	96%	99%
90%	90%	99%	100%

Table 2, Effects of Iteration

5. Remove the Opportunity for Error:

People are remarkable. We have the ability to adapt and solve problems that defies replication. However, people make mistakes. The only way to stop mistakes from happening is to remove the opportunity to make them. Detecting an error and fixing it is only half the job. The second half is determining the steps to make the error impossible to commit in the future. Normalizing the data is an excellent example of the technique.

6. Standard Interfaces:

Standard interfaces means creating strong generic interfaces between the parts of the system. Ease of integration is guaranteed for any piece that conforms to the standard. It allows an arbitrary number of pieces to be created in parallel with minimal interaction between the producers.

An example from our system is the coding standard that specifies how to create a software module. Other examples are building codes, electrical power, and mechanical fasteners.

7. Design for Change:

The only constant is change. Accept it and design processes, products, and organizations on the premise that today's optimal solution will not be tomorrow's. Specific attitudes and mechanisms are required for dealing with change. Systems designed on static assumptions will not have them.

Part of designing to accommodate change is understanding how to implement it. Jumping from the first floor of a building to the second is extremely difficult compared to walking up the stairs. In addition to the difficulty, leaping is also likely to be very disruptive with a long recovery time before a second may be attempted. The steps pose no such problem. In fact, with practice to build stamina, movement between floors may become continuous.

Organizations, processes, and products are no different. Little steps every day toward the desired goal will, in the end, allow for far more rapid and acceptable change than less frequent large changes.

8. Standardize, Optimize, Automate:

"Standardize, Optimize, Automate" is a prescription for how to create high-quality processes. The first step is to standardize on a process. It doesn't have to be perfect, just the best that can be developed in a reasonable time. (A good rule of thumb is whatever the experts can agree on within an afternoon.) Any process is better than no process, and the more people using a particular process, the faster its deficiencies will be discovered and corrected.

Optimization consists of using and correcting in a cyclic fashion. Optimization never really ends, but must occur before automation to prevent wasted effort. Automation means removing the potential for errors in the optimal process. Done properly, the results will be high productivity and quality.

A key to effective standardization is documentation. Formal documentation is useful but often impractical. Periodic formal updates augmented by a notebook of interim changes is very effective. Modifications can be added in the form of quick one-page updates, thereby reducing the burden of change.

9. Problem solving expected

Organizations have different cultures. One important aspect of that culture is the response to problems. The ideal is to create an atmosphere where everyone who encounters a problem reacts by trying to solve it. The worst occurs when people see problems and whine about them while

they wait for someone else to find the solution. Promoting the ideal response should be considered in every management response to difficulty.

One component of the response is assigning people who complain about problems to be responsible for solving them. At a minimum, they will never whine again because then they will be asked how the solution is progressing. At best they will feel empowered to take the initiative and do great things.

10. Make right easier than wrong:

The most important thing about the standards is that they must be implemented in a way that makes them easier to follow than not. Standard processes that double the time to accomplish a task are going to be circumvented by well-meaning engineers who think they are doing a good thing. When the process facilitates the work, people will follow it gladly. Every process has some unpleasant or boring work. Ideally it will be engineered into the normal process of doing the fun part.

Automation is a good way to achieve the desired effect. Lots of data collection can be done this way. If there is no way around the engineer actually doing the unpleasant work, then the fun work must be set up so that it cannot be completed unless the boring work is also done. This method has a pleasant side effect of motivating the engineer to figure out a way to automate or eliminate the undesired task.

Put checks in place that force required work to be completed before the process can be finished. (Do it now, because there'll never be time to go back.)

D. Teams:

In order to perform the work required to implement a feature, a systems engineer is required to write requirements documentation, a software engineer is required to implement the requirements in code, and a test engineer is required to verify that the functionality has been achieved. The work performed by one person on the feature must be reviewed and approved by the people who receive it.

To compress the development cycle and improve initial quality, each feature was assigned to a team consisting of one or more systems, software, and test engineers. The team as a whole is held responsible for the successful completion of the feature. They are expected to manage the work and report progress and delivery dates.

The idea is to create a shared destiny that forces the communication required to a) ensure a feature is specified so that it can be implemented and tested, and b) implemented according to specification in a way that can be verified, implemented, and tested right the first time.

E. PRCR System:

The lifeblood of the system put in place is the Problem Report Change Request (PRCR). Originally sheets of paper, the PRCR system has evolved into a fully electronic system for authorizing, tracking, and assigning work.

A PRCR starts with a problem description of change request. It may refer to other documentation for a further description. The author of the PRCR indicates to whom, they think, the problem belongs and sends it into the system. The PRCR is then brought to the Change Control Board (CCB) responsible for the designated area for review. From there the PRCR may be rejected, re-assigned to another CCB, assigned for investigation, assigned for implementation, etc. Wherever it goes, the PRCR is a permanent record of the problem and cannot be closed without a systematic evaluation.

A typical PRCR is written for implementing a new feature. It will be written by the responsible systems engineer who is assigned the PRCR to perform the systems work. When the systems engineer has finished that work, they attach a brief explanation of the work they accomplished and send it back to the CCB. The CCB may send it back for rework, but normally it is then assigned to the feature team's software engineer. The software engineer performs the required work, attaches a note explaining the results and sends it back to the CCB. The CCB then typically sends the PRCR to the feature team's test engineer to validate the successful resolution of the work. Again a brief note is attached, and it is sent back to the CCB for closure.

At any point in the process, the PRCR may be sent back to a previous step for rework. Often the software engineer will find specification problems that need to be corrected before it goes to test. In that case, the PRCR will go back to the feature team's system engineer and then on to the test engineer. The test engineer will regularly find feature problems, and the PRCR will cycle back to either software or systems for correction and then re-test.

F. Configuration Management:

An important feature of the PRCR system is that it is coupled with the configuration management system. All documentation, code source files, test plans, test results, etc. are stored in electronic format under version control. In order to make a change to the file, it must be "checked out for modification." Only a user with a valid PRCR assigned to them may perform that operation. Additionally, whenever a file is "checked back in" to the system, the PRCR database is updated to indicate which file and version number was modified.

The entire process is automated and flows very smoothly for anyone with the proper authorization. The real advantage is the tracking that is possible when a new person is added to the team or the work is transitioning between team members. It is easy to determine exactly what was and wasn't done for a particular PRCR.

G. Standards:

In the new process everything was developed to a standard. Code, requirements, and testing all were standardized. The purpose, as with most standards, was to spread best practices, facilitate communication, and speed up the work. The key is to have living standards. With everybody using the same techniques, flaws are rapidly discovered, new solutions found and the processes updated for the next go-around.

One especially powerful result of the standard work is the interchangability of team members. System, software, and test engineers can be moved from team to team without any training

penalty. The entire team becomes much less sensitive to the normal disruptions of turnover and promotions.

The increased speed of the work cannot be over-emphasized. Without standard practices, a large part of the engineering effort is dedicated to figuring out document formats, explaining diagram symbols to confused coworkers, deciding what to name variables, etc. The real value-adding efforts must compete for the engineer's time. With good standards, the percentage of time available for truly useful work is easily 75%.

H. Integration:

Software integration, in the traditional sense, has been eliminated. The basic function is still performed, but the implementation has been modified. The main reason for deviating from the normal practice of "design, code, integrate" is that it doesn't work. Typical integration phases are open-ended and unpredictable. The only constants are that integration will take longer than expected and the integrators will have a miserable experience.

The problem lies both in when and who performs the integration. Integration has several functions. One that everybody recognizes is putting all the pieces together into a finished product. More importantly, integration validates interfaces. Delaying the validation until the end of the development cycle hides problems that need to be resolved much earlier. Additionally, the number of problems introduced simultaneously makes them virtually impossible to identify. Good practice indicates that changes should be introduced one at a time so that cause and effect are easier to determine, which is violated by the very nature of the process.

The problem of who does the integration is even worse. The integrators are, at best, a subset of the people who created the material to be integrated. They have little familiarity with the changes and no time spent considering the ramifications. When two conflicting modifications are integrated, they have to recreate the thought processes of the original implementors of each change. Even when they are the original implementors, the time between creation and integration fades a lot of memory.

The solution is to perform incremental integration [3]. A development cycle starts with a baseline. The first change completed is immediately integrated into the baseline by its implementor. The new developmental baseline is then available, with a guarantee of functionality, for others to integrate their changes. Then the process is repeated. The good things accomplished by the incremental method are the following:

- The minimum set of changes is implemented at any one time.
- All of the information about the change is known by the integrator, and it is still fresh.
- Conflicting changes are detected as soon as possible, leaving maximum time for resolution.
- Low quality changes and integration are quickly, visibly, and traceably detected.
- The developmental baseline can be turned into a release on any given day.

I. Peer Pressure:

Motivating people to perform at their best is often difficult. One of the most powerful tools available is peer pressure. It is used implicitly and explicitly to great advantage. All processes were designed, where possible, to be visible. Take, for example, the review process for specifications, incremental integration for software, and PRCR tracking. All promote an environment where people's work, both quantity and quality, is readily apparent. Reviews and integration provide short-term accountability and the PRCR system provides long-term accountability.

Explicitly, we used weekly status meetings with all members of systems, software, and test teams present. We would review the current progress and plan future deliverables. Each team that was assigned a feature was required to commit to a completion date for the work. The process was intentionally run in a non-confrontational manner. The only comments were to note success or congratulations. Publicly requested schedule slips were always granted. If a slip was a problem, it would be noted as something we would work out over the coming week. Any negative feedback was always given in private.

Both types of peer pressure seem to be very effective. They are very powerful and potentially sources of abuse if used improperly. Especially in status meetings, great sensitivity must be used by the leaders of the meeting to keep the mood upbeat and friendly. No team member should feel threatened or treated disrespectfully.

IV. Cell Model:

A. Basic Modeling Decisions

Modeling a lean manufacturing system is, potentially, a broad undertaking. Many types, sizes, and variations have been developed over the years. The most common one that comes to mind is automobile assembly as practiced by Toyota. While interesting and worthy of study, it is too large to be studied easily. Fortunately, the basic concepts, applied on a large scale to a vehicle assembly line, also apply on a small scale, to a relatively simple assembly cell.

I have created a model of a basic assembly cell as commonly discussed in lean manufacturing [1] [10-13] [16] [17]. Using it as a base, I have then added the basic loops of the lean manufacturing process, specifically error-proofing and process improvement. The rest of this section provides a conceptual description of the model. Appendix A includes a detailed listing of the modeling equations. Figure 1 shows the basic causal loop diagram from which the model was developed.

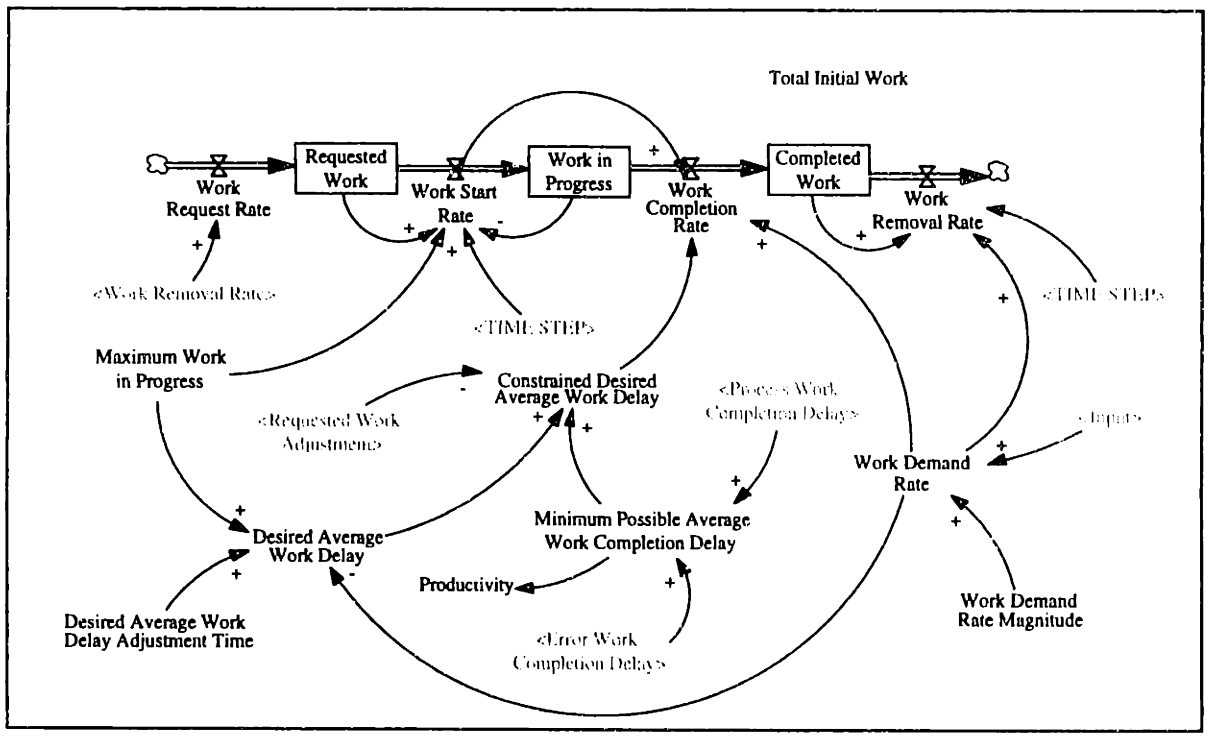

Figure 1, Work Cell

B. Work Cell

The work cell is the heart of this model. It contains two sections. The first is the cell itself, consisting of *Requested Work*, *Work in Progress*, and *Completed Work*. The second is the computation of the *Constrained Desired Average Work Delay*.

The cell implements a standard kanban system for the work. Each piece of *Completed Work* is a widget with a kanban card attached. Removing work takes the widget out of the stock of completed work and places the kanban card representing the widget into the stock of *Requested*

Work. Work in Progress is a standard third-order material delay where *Work Start Rate* is limited by the availability of *Requested Work.*

The delay used in the system is an aggregation of the effects of productivity (widgets/week) and quality (percent good widgets). While they could be modeled separately, their effects combine nicely into a simple delay. The delay due to productivity is simply the reciprocal of the productivity. The delay due to quality is created by the extra time required to produce widgets that are scrapped. For quality of 50%, the straight productivity delay would be doubled, since, on average, a good widget is produced only every other time.

The *Constrained Desired Average Work Delay* models the desire in a lean production system to have the cell maintain a constant WIP by setting the work rate (delay) to match the demand rate. The *Constrained Desired Average Work Delay* is adjusted to match the demand rate. In an actual assembly cell, the adjustment would be accomplished by adding or removing workers to the cell. (The demand rate is set by the takt time.)

Minimum Possible Average Work Completion Delay is the best possible performance of the cell. It is the key performance parameter in the model. Since delay time is a counter-intuitive measure, I have added its reciprocal, *Productivity,* to the model to use in presenting data for the analysis.

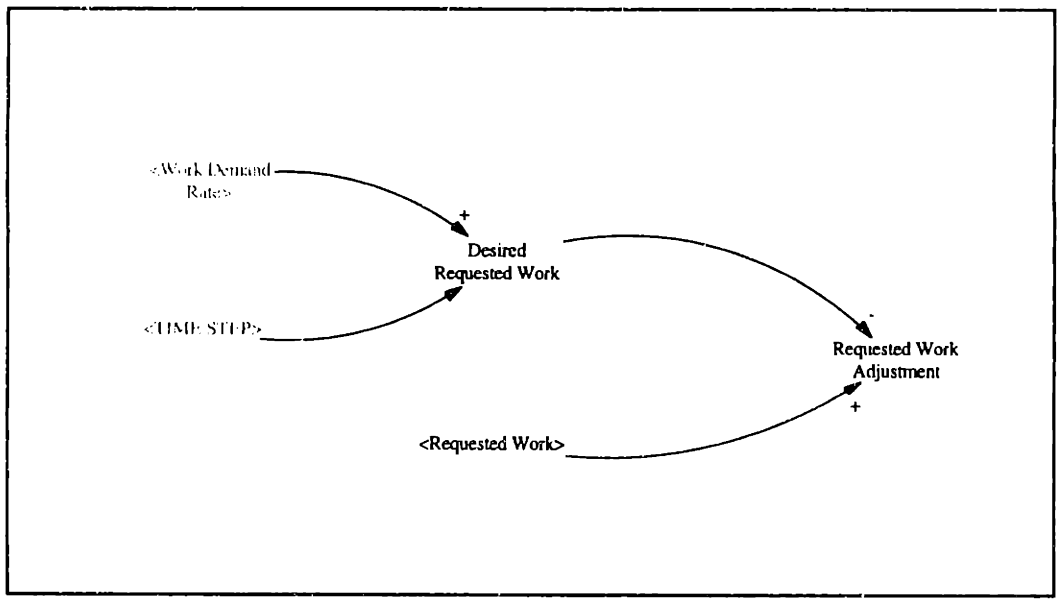

Figure 2, Work Cell (cont...)

Requested Work Adjustment modifies *Constrained Desired Average Work Delay* when *Requested Work* is greater than required to match the *Work Demand Rate.* Once the required steady state is achieved, there may still be excess requested work. In order to catch up, the cell must be operated above the demand rate.

C. Jidoka (Potential Error Removal)

The jidoka section of the model is straight forward and consists of three parts. The first part is the stock of *Potential Errors.* It is initialized with the total number of production errors that can be eliminated by jidoka. As the *Potential Errors* are discovered and removed, the stock is reduced at

the *Jidoka Implementation Rate* calculated from the *Error Elimination Half Life* [7]. The *Jidoka Implementation Rate* is multiplied by the *Effectiveness of Improvement Process*, which is a percentage. Thus, for a perfect improvement process, the *Potential Errors* decrease at the maximum possible rate. For a completely ineffective process, *Jidoka Implementation Rate* is zero.

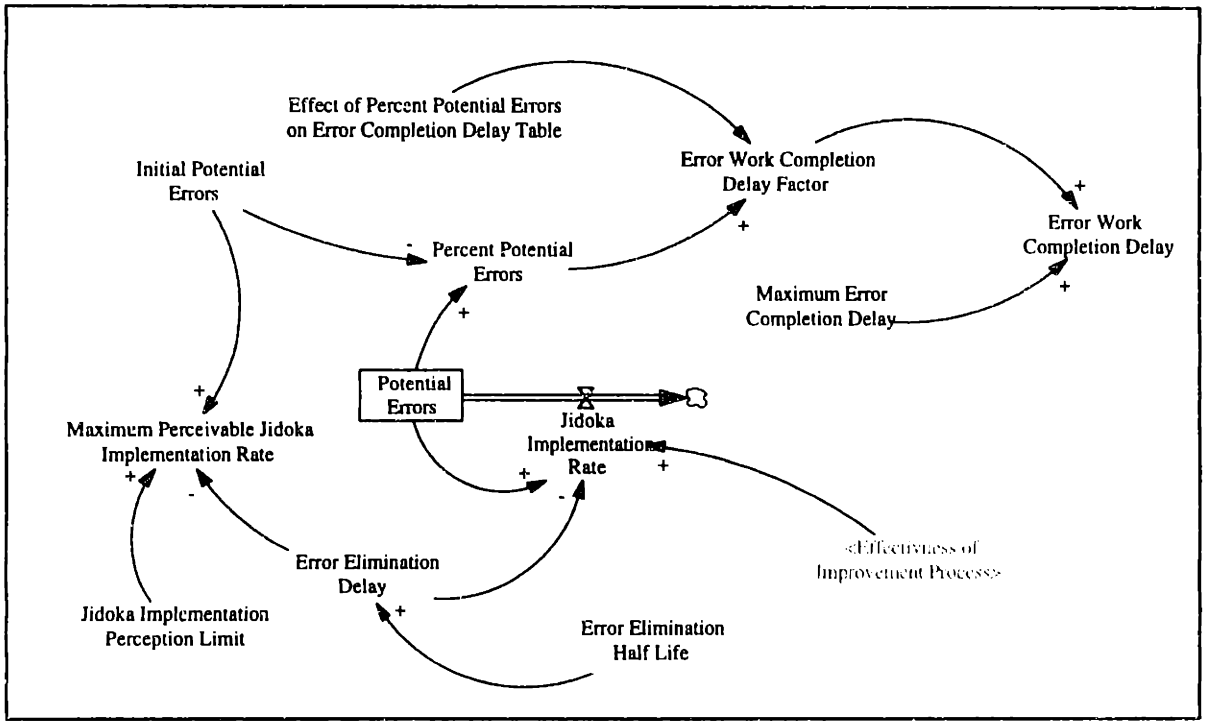

Figure 3, Jidoka (Potential Error Removal)

Error Work Completion Delay is calculated from the percentage of potential errors left in the cell. Initial reductions in *Potential Errors* have a small effect, since there are so many others. Elimination of the last few potential assembly errors also has little effect, because they are so rare at that point. The result is an S-shaped curve for the relationship between *Percent Potential Errors* and *Error Work Completion Delay*. (The curve is normalized so that it can be scaled with *Maximum Error Completion Delay*.)

The last section is the calculation of *Maximum Perceivable Jidoka Implementation Rate*. It represents the limited ability of workers to perceive the *Jidoka Implementation Rate*. Beyond the *Jidoka Implementation Perception Limit*, any higher *Jidoka Implementation Rate* is indistinguishable from the limit. *Maximum Perceivable Jidoka Implementation Rate* is used in the motivation section of model.

D. Process Delay

The process delay section of the model is similar to the jidoka section, except that the stock, *Process Work Completion Delay*, not only decreases at *the Process Improvement Rate*, but increases due to *Process Entropy*. Additionally, the delay due to the practice of the cell process by the workers is modeled directly instead of the number of mistakes. It is limited from shrinking below the *Minimum Process Work Completion Delay*, which represents the smallest delay

possible with the perfect practice of the process. The concepts behind the half-lives and *Maximum Perceivable Process Improvement Success* are the same.

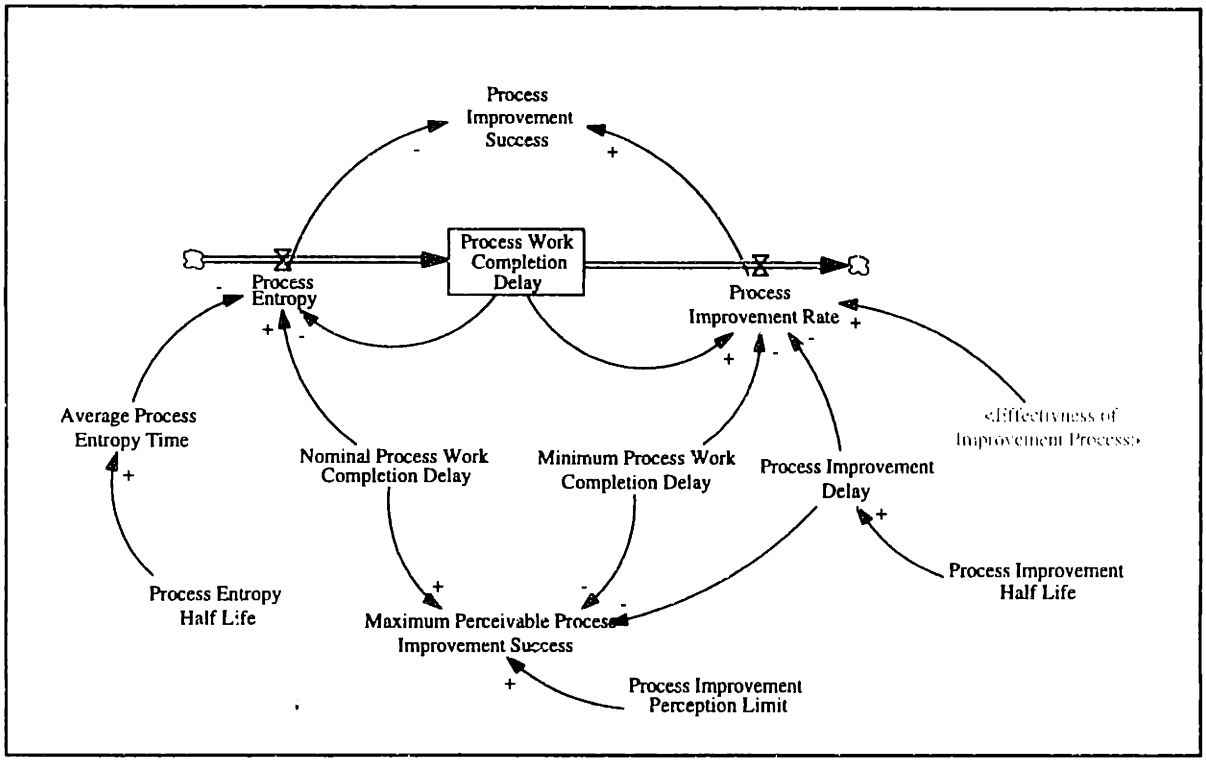

Figure 4, Process Work Completion Delay

Process Entropy comes from the idea that there is a *Nominal Process Work Completion Delay* towards which the cell will tend over time. It represents the natural state of the cell process if the workers are left to their own devices. *Process Entropy* is the rate at which this happens. *Process Entropy Half Life* is assumed to be smaller than *Process Improvement Half Life*. This is because, improvements tend to be quick compared to the gradual decay of a system.

E. Motivation

Motivation is a key aspect in this model. The basic improvement loops are driven by the desire of the workers to engage in improvement activities. Although often forgotten in western implementations of lean manufacturing, kaizen activities designed to promote and reward participation in process improvements are very important. I have not attempted to model the actual motivational process. In the interest of simplicity, I have adopted a *Nominal Motivational Level,* which represents the basic motivation of the workforce. Activities that raise or lower it are considered exogenous. Included in the model are modifiers to the basic motivational level.

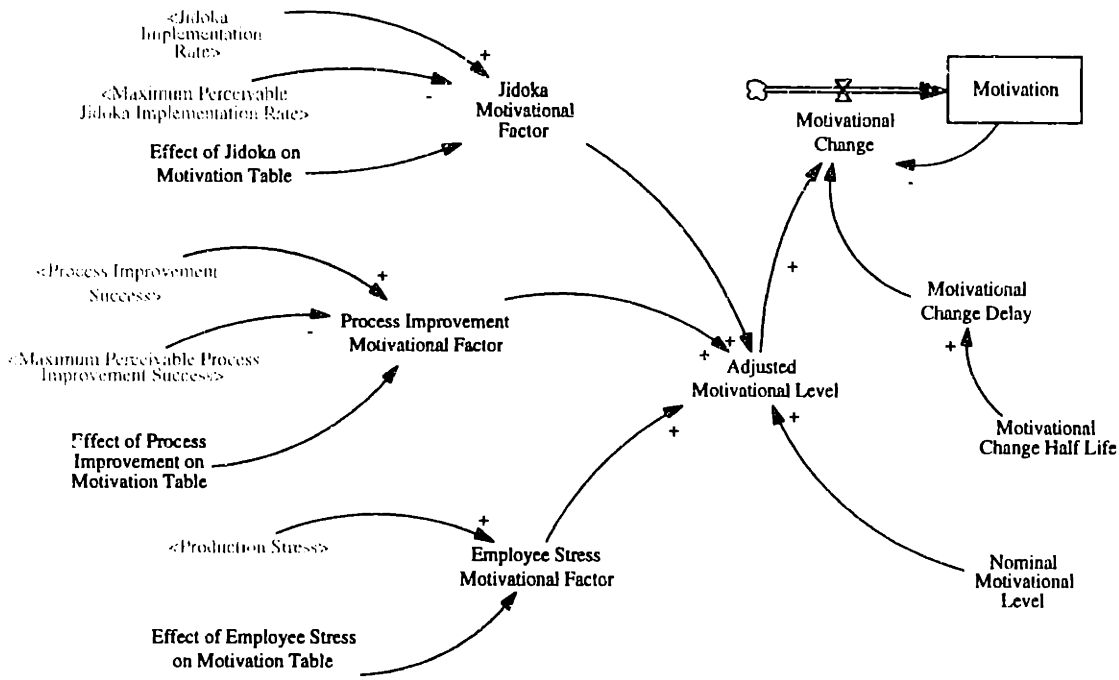

Figure 5, Motivation

Jidoka Motivational Factor, *Process Improvement Motivational Factor*, and *Employee Stress Motivational Factor* modify the *Nominal Motivational Level* to obtain the *Adjusted Motivational Level*. The actual *Motivation* is a stock that tends toward the *Adjusted Motivational Level* based on the *Motivational Change Half Life*. This structure is used to model the lag in average motivation caused by the time it takes for diffusion of motivational factors across an organization. With good kaizen activities, the half-life will be very short. Without them it can be very long.

Jidoka Motivational Factor is derived from the ratio of the actual *Jidoka Implementation Rate* and the *Maximum Perceivable Jidoka Implementation Rate* using a table function. The table is an S-shaped curve, since small success rates generate little enthusiasm, and after a certain point, a higher rate cannot generate much more enthusiasm. A ratio of zero generates a factor of one, and above a ratio of 1.5 the factor is two. *Jidoka Motivational Factor* is never less than one, thereby reducing the *Adjusted Motivational Level*, because the *Jidoka Implementation Rate* is never negative. Since the number of *Potential Errors* can never increase, there is no way for it to be demotivating.

Process Improvement Motivational Factor is similar, except it can be demotivating as well as motivating. Figure 6 shows the relationship between the rate of implementation of process improvement and worker motivation. Since loss of improvement is as demotivating as improvement is motivating, there the result is two S-shaped curves connected with an inflection point at (1, 1).

Jidoka Motivational Factor and *Process Improvement Motivational Factor* result from worker measurement of the system. There are many choices for how to measure improvement success or failure and feed it back to the workers. I have chosen to model the simplest method, where nothing special is done and the workers' own perceptions are all that indicate the improvement

rates. Other methods exist and could be included in the model to determine their effects. Such an effort is beyond the scope of this investigation. (I speculate that different measurement methods will only vary in their sensitivity and duration of effect.)

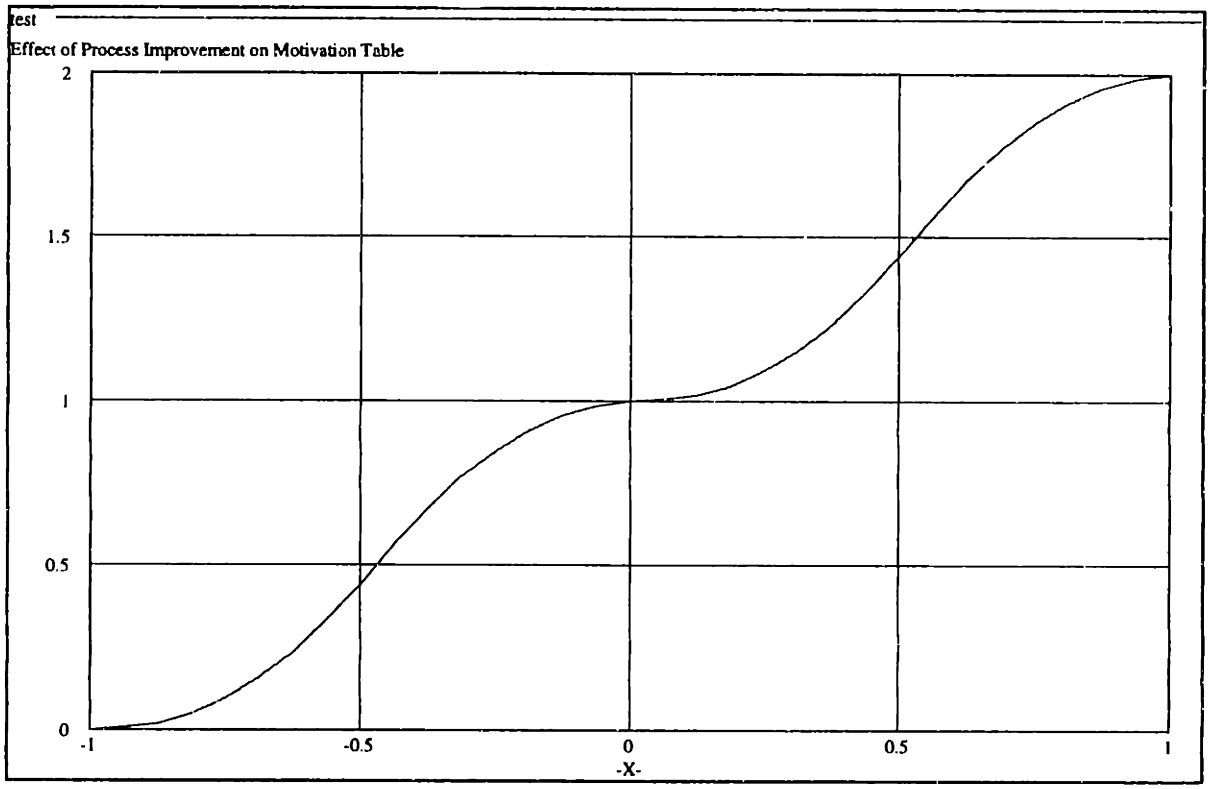

Figure 6, Effect of Process Improvement on Motivation Table

Employee Stress Motivational Factor models the effect of the state of the assembly cell on worker motivation. It is only a positive motivational factor because it is a result of the measurement of the state of the cell, not worker fatigue or burnout. The reasoning is that as the cell gets behind, it will initially motivate workers to improve. However, if the cell stays behind, they will not be demotivated by the state of the cell as their goals erode.

F. Production Stress

Production Stress is the result of feedback from the kanban system to the workers. When the cell is meeting the *Work Demand Rate,* the majority of the work will be in the stocks of *Work In Progress* or *Completed Work.* When the current capacity of the cell is exceeded, the work in the cell, which is limited by the number of kanban cards, will shift to the requested work stock. The movement of the kanban cards makes this very visible, and produces the *Cell Production Stress* based on the *Percent Requested Work.*

Effect of Percent Requested Work on Cell Production Stress Table represents the relationship between *Percent Requested Work* and *Cell Production Stress.* A small backlog produces a small amount of stress, while higher backlogs rapidly produce the maximum level of stress, resulting in a very sharp S-shaped curve.

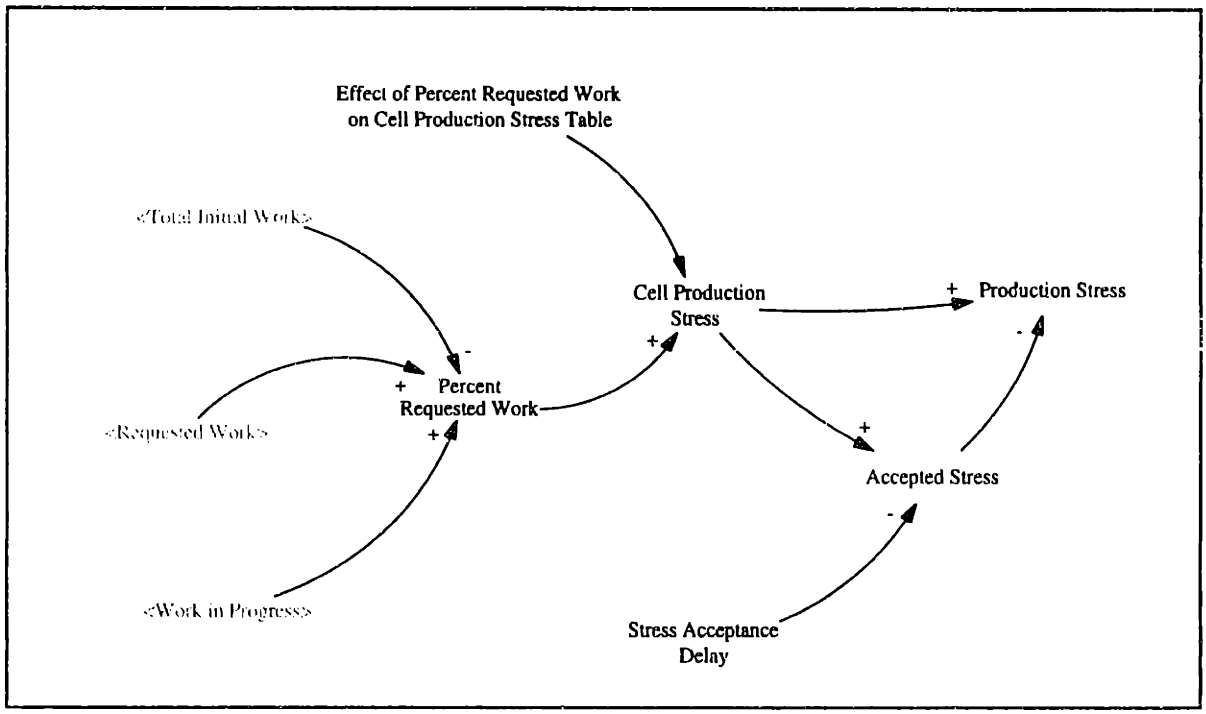

Figure 7, Production Stress

Production Stress is different from *Cell Production Stress* because of the effect of eroding goals. Although the kanban system will always indicate the state of the system, unless the system is able to respond to the stress, the workers will eventually begin to accept the excess requested work situation as normal. As *Cell Production Stress* persists, the level *Accepted Stress* increases until *Production Stress* falls to zero. (*Accepted Stress* is modeled as a third-order information delay of *Cell Production Stress.*)

G. Improvement Process

I have modeled the improvement processes very simply. The exact process used and its effectiveness is represented by *Quality of Improvement Process* and *Improvement Process Usage*. *Quality of Improvement Process* is treated as an input parameter with a value of one representing a perfect process and a value of zero representing a useless process. Studying different processes and their relative quality is a good area for further investigation.

Usage is determined from *Motivation* and a table. *Effect of Motivation On Improvement Process Usage Table* represents the relationship between worker motivation and improvement process usage. Below a certain threshold of motivation, the improvement process will not be used; while above the minimum threshold, usage rapidly becomes 100%. The result is a very sharp S-shaped curve. The table is calibrated so that a motivation of one-half is just starting to cause usage of the improvement process.

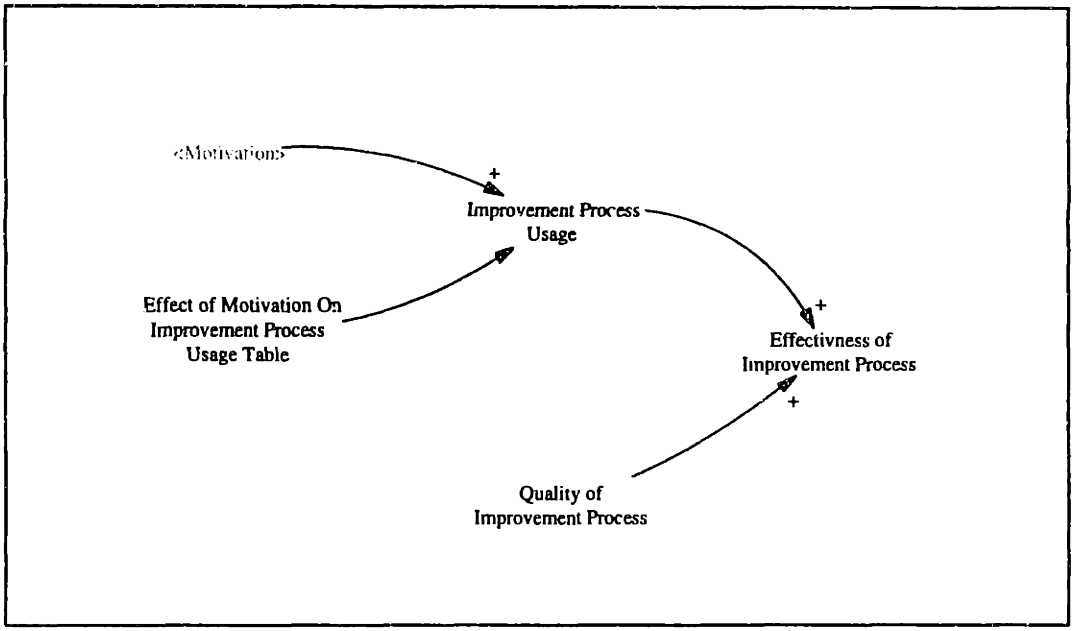

Figure 8, Improvement Process

H. Signal Generator

The signal generator is a small utility originally developed by Prof. Repenning as an easy way to generate a variety of input signals for a model. Please see the appendix for details on the signal generator equations.

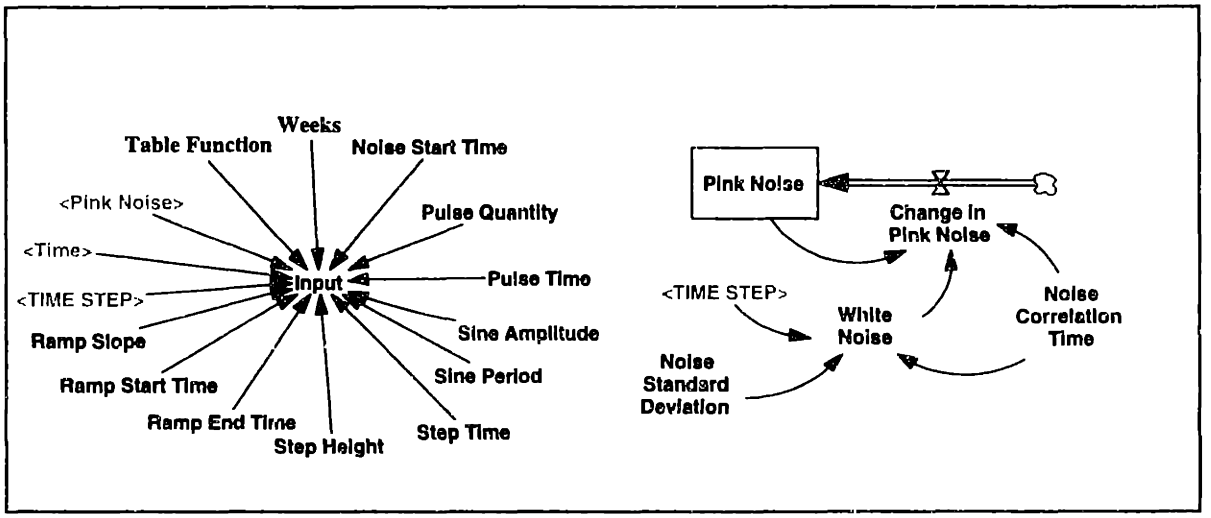

Figure 9, Signal Generator

V. Cell Model Analysis:

A. General Results

With the current formulation, the cell model is controlled by the inventory, demand rate, quality of improvement process, employee motivation, and system measurement. By varying these input parameters, the various modes of behavior resulting from the usage of a lean production system can be replicated. The modes presented here are success, failure, and replication of the effects of inventory [12]. In addition to replication of the reference modes, the general effects of the basic policy levers is examined. The effects of motivation, improvement process, and measurement sensitivity are examined.

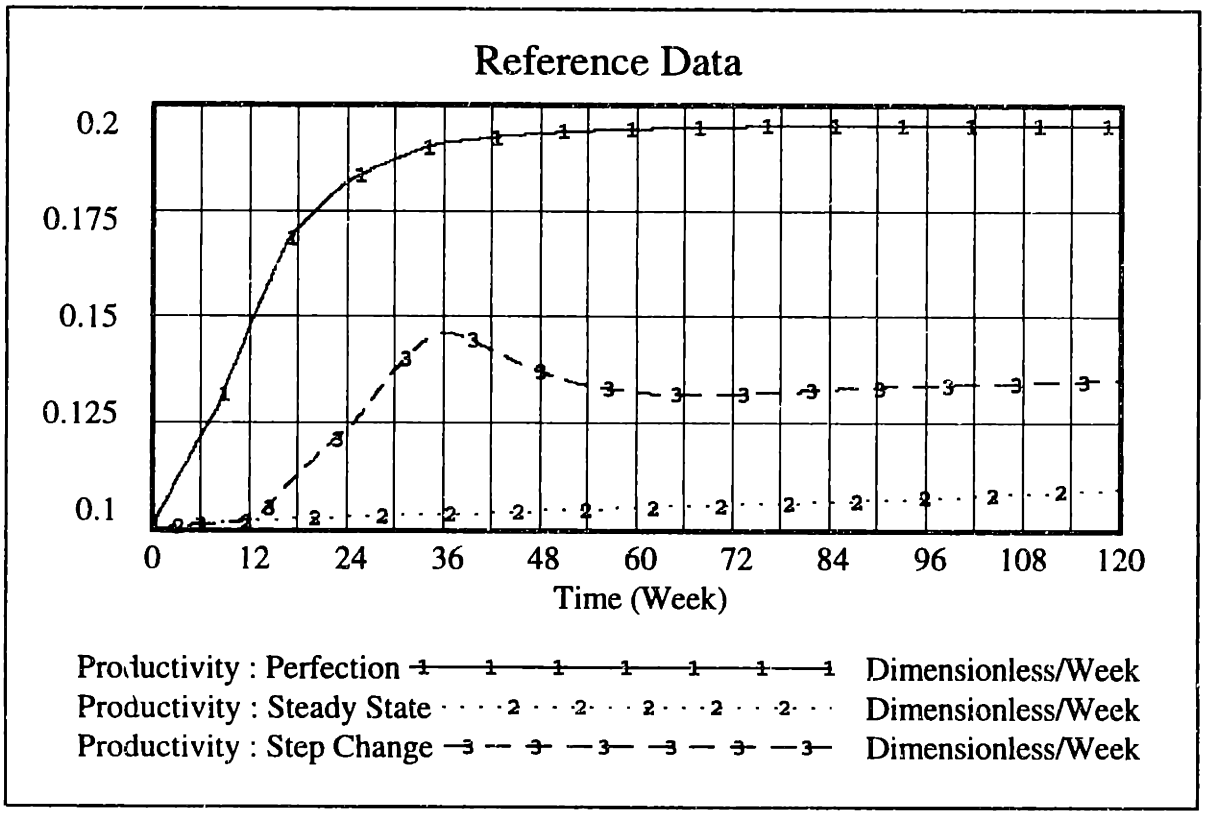

Figure 10, Reference Data

In analyzing the model, the important parameters are the *Work Demand Rate* and the *Productivity*, (1 / *Minimum Possible Average Work Completion Delay*). *Work Demand Rate* drives the system as the only dynamic input. For all of my analysis, it is either a constant or that constant times a step function, which has a magnitude increase of 50% at week five. *Productivity* summarizes the capabilities of the system. It is the sum of the effects of error prevention and process improvement and is the upper limit of the productivity of the cell.

To clarify the results, three reference data sets have been created with the nominal values of the input parameters as shown in Table 3. The first is the system in steady state. It defines the lower limit of the desired system performance. The second is identical to the steady state case with a

step change in *Work Demand Rate* to stimulate the system. (The increase is such that productivity must increase from 0.1 to 0.15 to regain steady state.)

Parameter	Nominal Value	Minimum	Maximum
Nominal Motivational Level	0.5	0.0	1.0
Quality of Improvement Process	0.5	0.0	1.0
Process Improvement Perception Limit	0.7	0.01	1.0
Jidoka Implementation Perception Limit	0.7	0.01	1.0
Total Initial Work	7.0	1.0	N/A

Table 3, Parameter Settings

It defines the median improvement performance of the system. The third is perfection where a step change is applied with *Nominal Motivational Level* and *Quality of Improvement Process* set at the highest possible level. It defines the upper performance limit of the system. All subsequent data are presented in this context.

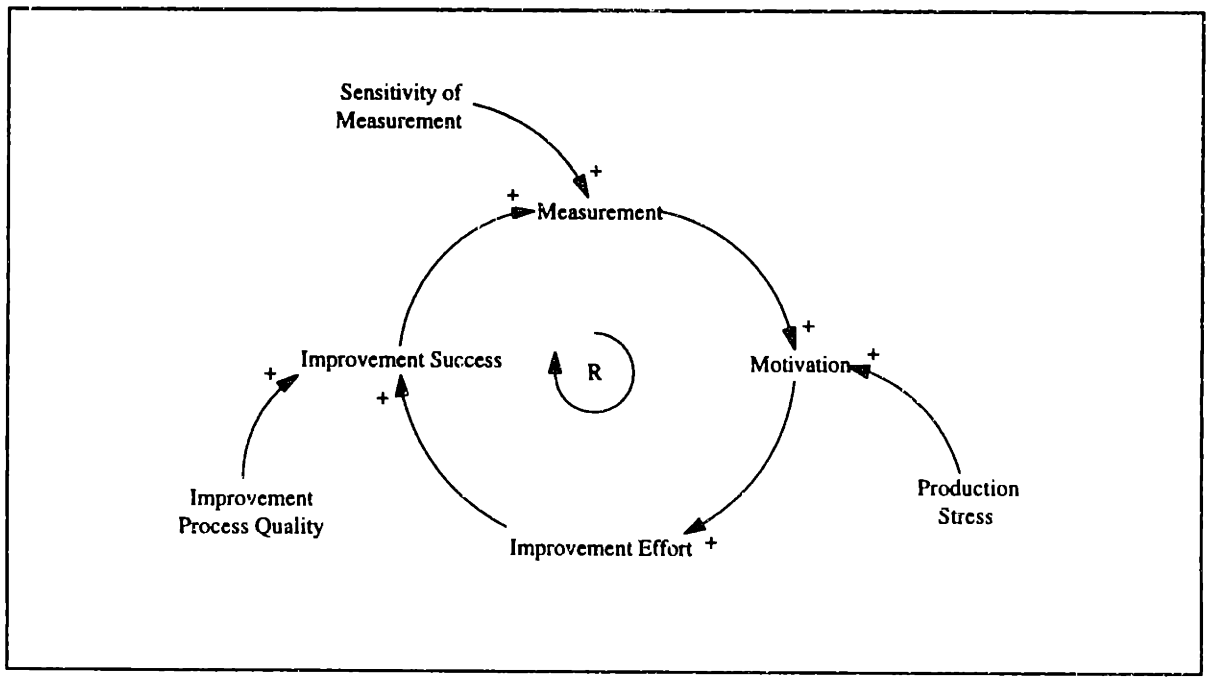

Figure 11, Simplified Causal Loop

Figure 11 shows a simplified causal loop diagram for the system that helps clarify how the cell model works. (The process and jidoka loops have been agragated into one imprrovment loop.) The system is designed as a positive reinforcing loop with motivation, improvement effort, improvement success, and measurement. Improvement process quality and sensitivity of measurement control the strength of the loop. If they are low, the loop is effectivly disabled. Increasing them increses the strength of the loop. Lean production mechanisms (kanban, etc.) translate the state of the assembly cell into production stress, which drives the loop.

The reference data are excellent examples of how the loop works. In the steady state case, the loop is barely functioning. There is some improvement from the nominal level of motivation, but the improvement process quality and measurement sensitivity aren't high enough for the loop to start itself. In the perfection case, the loop is self-starting. It quickly drives the assembly process to the optimal limits and keeps it there.

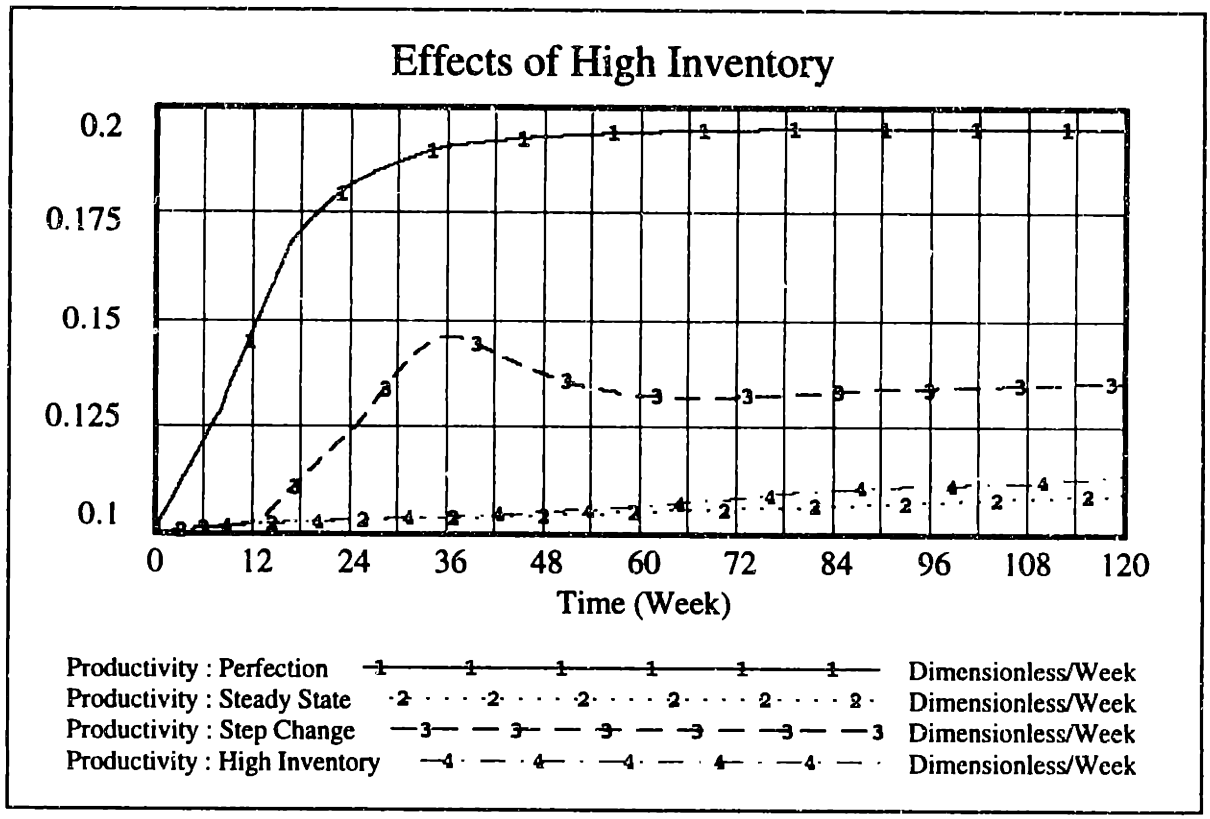

Figure 12, Effects of High Inventory

The step change case is particularly interesting. The jump in demand provides enough of a kick that the loop starts reinforcing itself. Then, as the production stress is removed, the loop shuts down. Productivity decays due to entropy until the steady state condition is reestablished. The only reason the cell doesn't fall back to the old level of productivity is because of the jidoka improvments.

It is important to note that there are two ways in which the production stress can be removed: a) the cell catches up with demand or b) the stress is accepted and stops being a motivating factor. In the step change case, the cell doesn't meet demand because the improvement process quality limits the growth in productivity until the stress becomes acceptable. In the case where the cell is able to catch up with demand, productivity exceeds demand for a time and then falls back to match demand. At that point, the lean manufacturing system acts like a regulator to keep productivity matched with demand. The jidoka improvement is permanant, but the process improvement begins to decay with entropy when stress is removed. As productivity falls below demand, production stress is produced, which drives improvement efforts until the cell catches up. Then entropy takes over and the process repeats itself.

B. Inventory

One of the important modes of the system is demonstrating the effects of inventory. One of the clearest points in the literature about lean production is that high levels of inventory will mask the true state of the system and prevent improvement. Any model of a lean production cell must be able to replicate the effects of inventory.

To test the cell model, I ran the Step Change reference case with Total Initial Work increased to 30. The resulting data is shown in Figure 12. Clearly the excess inventory "turns off " the system improvement.

C. Success and Failure

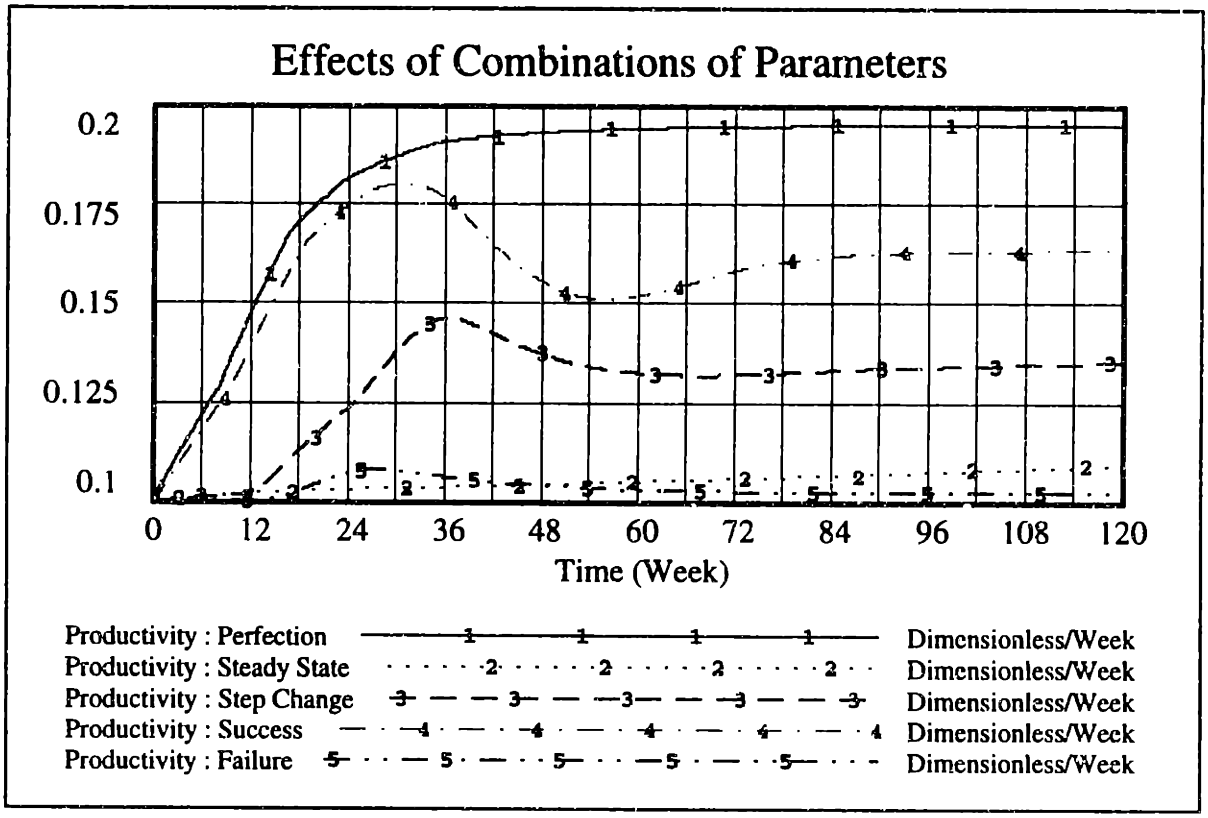

Figure 13, Effects of Combinations of Parameters

Typically, two types of stories come from companies that attempt to implement lean manufacturing techniques. The first are the success stories: "We put the system in place and it works great. We've doubled our quality and productivity."

The second type are the failures: "Everything just fell apart. We're worse off now than we were before." [6] [11] As shown in Figure 13, the model is capable of generating both reference modes. Table 4 shows the selection of input parameters for each mode.

Parameter	Success	Failure
Nominal Motivational Level	0.6	0.4
Quality of Improvement Process	0.9	0.4
Process Improvement Perception Limit	0.5	1.0
Jidoka Implementation Perception Limit	0.5	1.0

Table 4, Parameter Settings for Success and Failure

D. Oscillations

One of the interesting aspects of the model is the damped oscillation produced in response to the step change in *Work Demand Rate*. The sources of the oscillation can be seen by examining the inputs to *Minimum Possible Average Work Completion Delay*. It is the sum of the delay caused by potential errors and the work process. As shown in Figure 14, the delay due to potential errors has no oscillation. It comes from the work process delay.

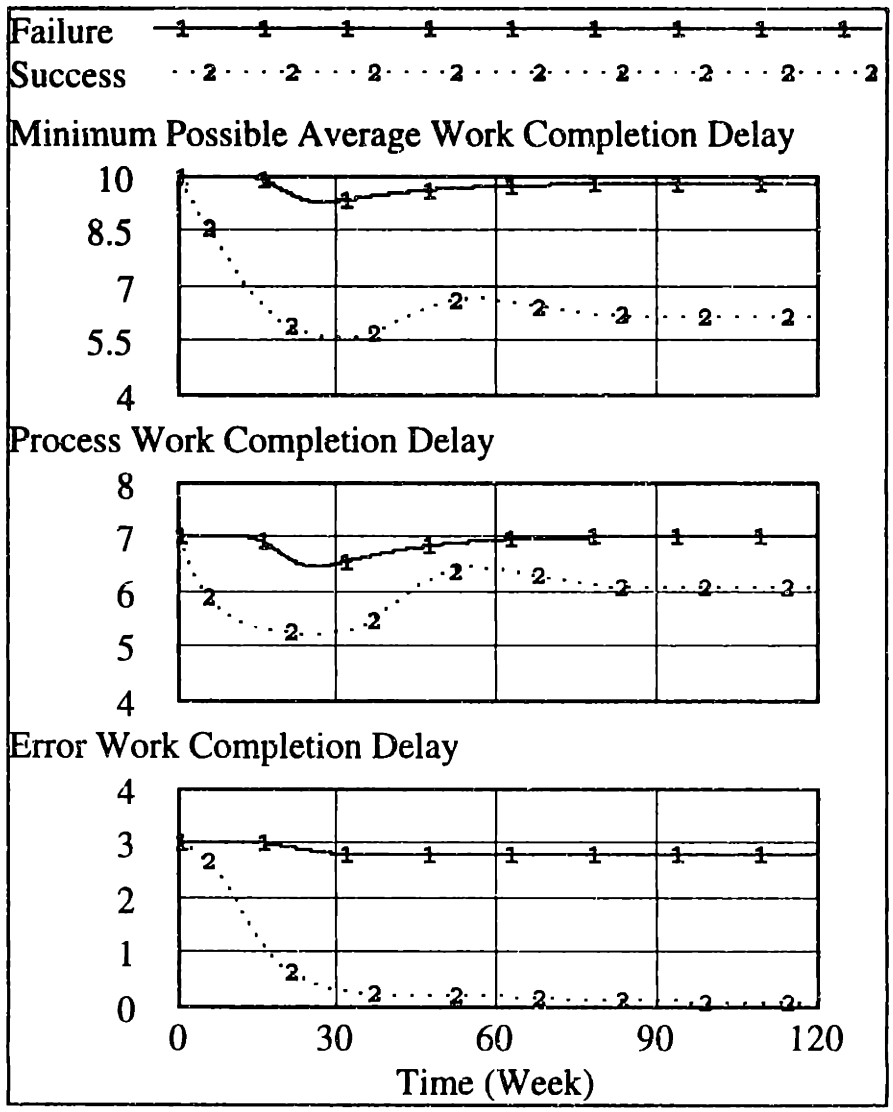

Figure 14, Sources of Oscillation

Figure 15 clearly shows the reason for the oscillations. While the *Minimum Possible Average Work Completion Delay* is the potential performance of the cell, *Constrained Desired Average Work Delay* is the actual performance of the cell. The steps in actual performance correspond to the initial step in *Work*

Demand Rate and the subsequent return to steady state when the cell worked off all the excess *Requested Work.*

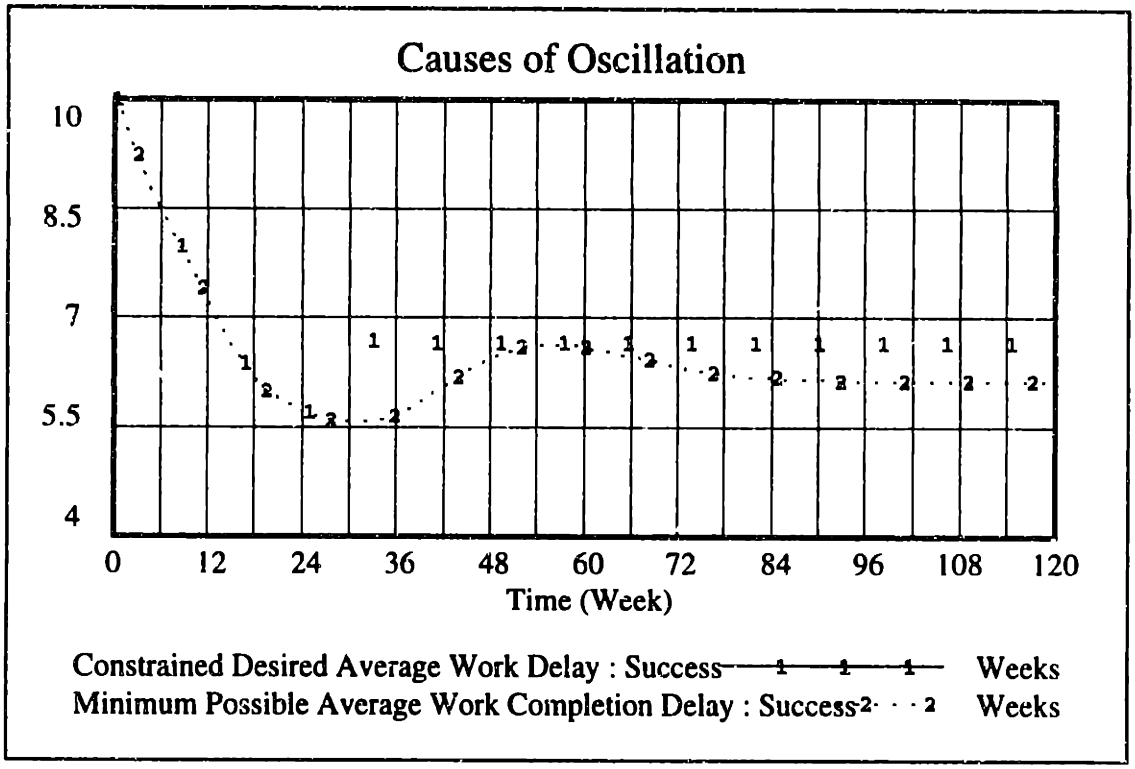

Figure 15, Causes of Oscillation

After the backlog is removed, the work process stops improving. Entropy takes over, and the process delay begins to increase. This trend continues until the ability to meet the current demand rate is threatened and improvement is started up again. At that point the combination of the jidoka improvements and the decrease in staffing take over to lock in enough improvements that the oscillations are damped out.

E. Motivation

The effects of varying *Nominal Motivational Level* while leaving the rest of the parameters at their nominal value is shown in Figure 16. The values from the success and failure modes above were used. Worker motivation plays an important role in a successful implementation of lean manufacturing. Notice that simply increasing the motivational level enables the cell to meet the increase in demand. Even without the step change in demand, *Productivity* begins increasing. In the nominal and low motivation cases, the increase in demand is required to get the improvement processes started.

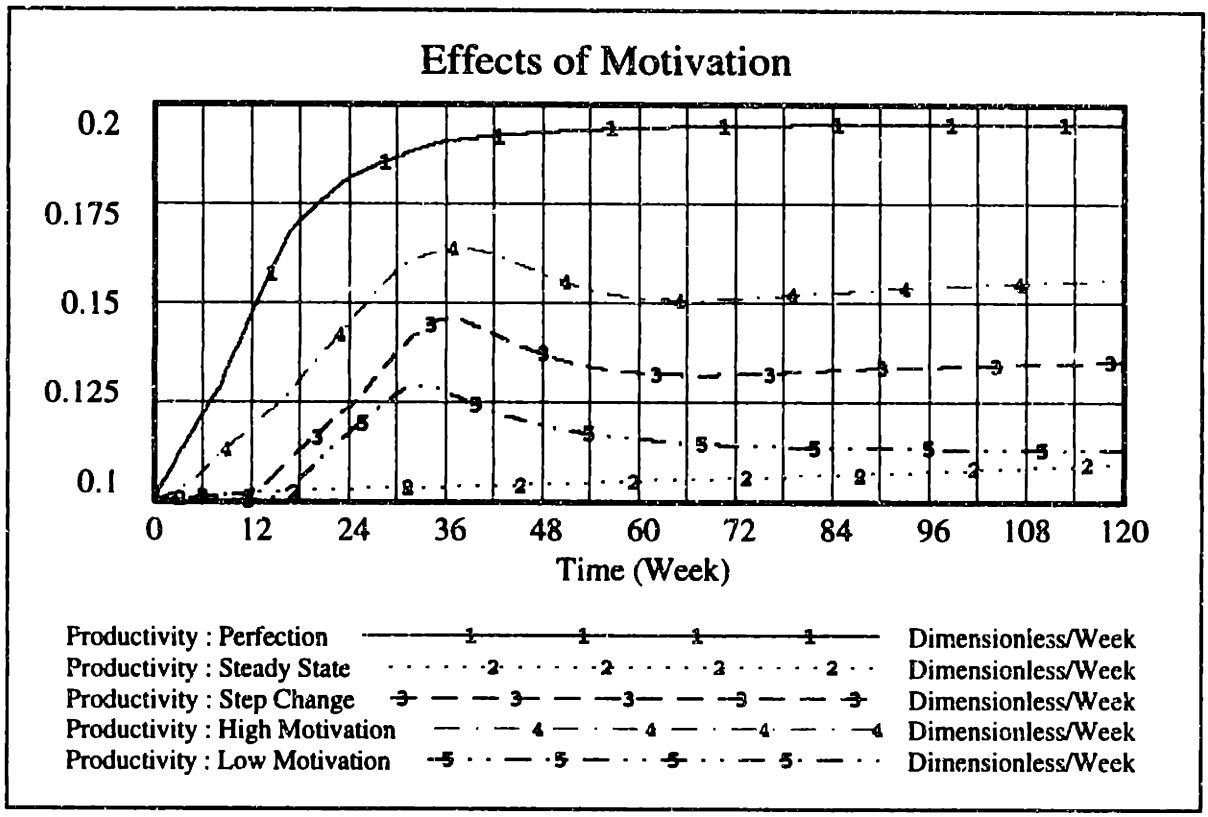

Figure 16, Effects of Motivation

F. Improvement Process

The effects of varying *Quality of Improvement Process* while leaving the rest of the parameters at their nominal value is shown in Figure 17. The values from the success and failure modes above were used. The improvement process primarily affects the height of the peaks in *Productivity*. Although the final value of *Productivity* is increased, the improvement process is not sufficient to meet the increase in demand on its own. After good initial success, the final level of productivity falls below the 0.15 required to meet the increase in demand.

G. Measurement Sensitivity

The effects of varying measurement sensitivity, *Process Improvement Perception Limit and Jidoka Implementation Perception Limit*, while leaving the rest of the parameters at their nominal value is shown in Figure 18. The values from the Success and Failure modes above were used. Similar to the improvement process, measurement sensitivity primarily affects the height of the peaks in *Productivity*. The effect is greatly diminished and the final value of *Productivity* is virtually unchanged.

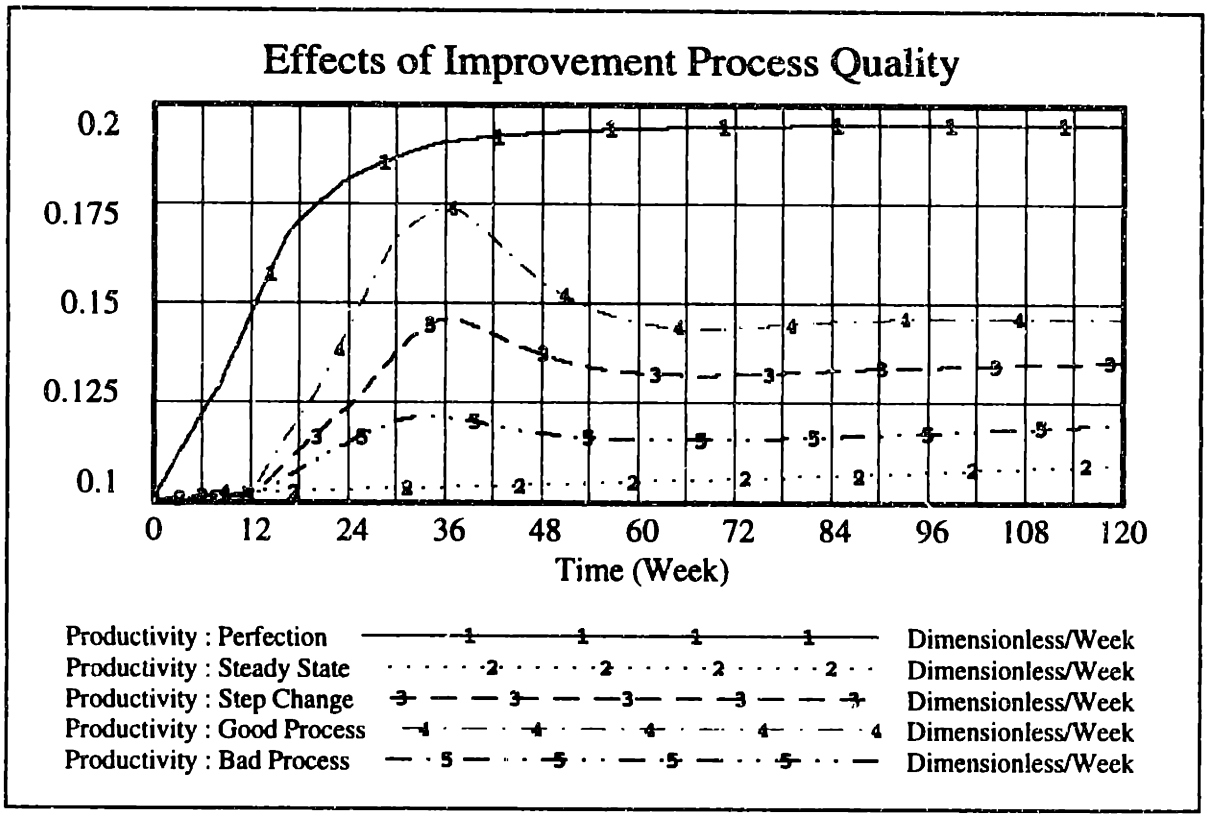

Figure 17, Effects of Improvement Process

H. Conclusions

I have created a workable model of a lean manufacturing cell. By building the basic structure prescribed in the literature, I have been able to replicate several widely reported reference modes. In the process I have gained several valuable insights into the system.

Overall the basic system works. The structure of a lean manufacturing system has two very strong reinforcing loops. Everything is designed to steam-roll toward the desired outcome. The kanban system is designed to be very sensitive, providing feedback when problems arise. Every problem is designed to be amplified to drive the improvement processes. Properly harnessed, the information can be used to drive toward higher productivity and quality. Improperly used it is merely a source of stress and disruption.

There are two improvement mechanisms, of which the most powerful is the jidoka loop. Since the improvements from it are built into the system, once they are found they are never lost. This makes jidoka easy to implement.

The process improvement loop is also very powerful but also potentially transient in nature. Since people are involved, the level to which it is practiced is dependent on the demands made on the system. It explains why practitioners of lean production suggest setting the takt time slightly below the ability of the production line to deliver. With the proper motivation in place, over-stressing the system provides the incentive required to develop process improvements. It also explains the desire to match the production of the cell to the demand rate. By staffing the cell at

the minimum required level, no slack time is created in which the practice of the process can deteriorate without immediate negative feedback.

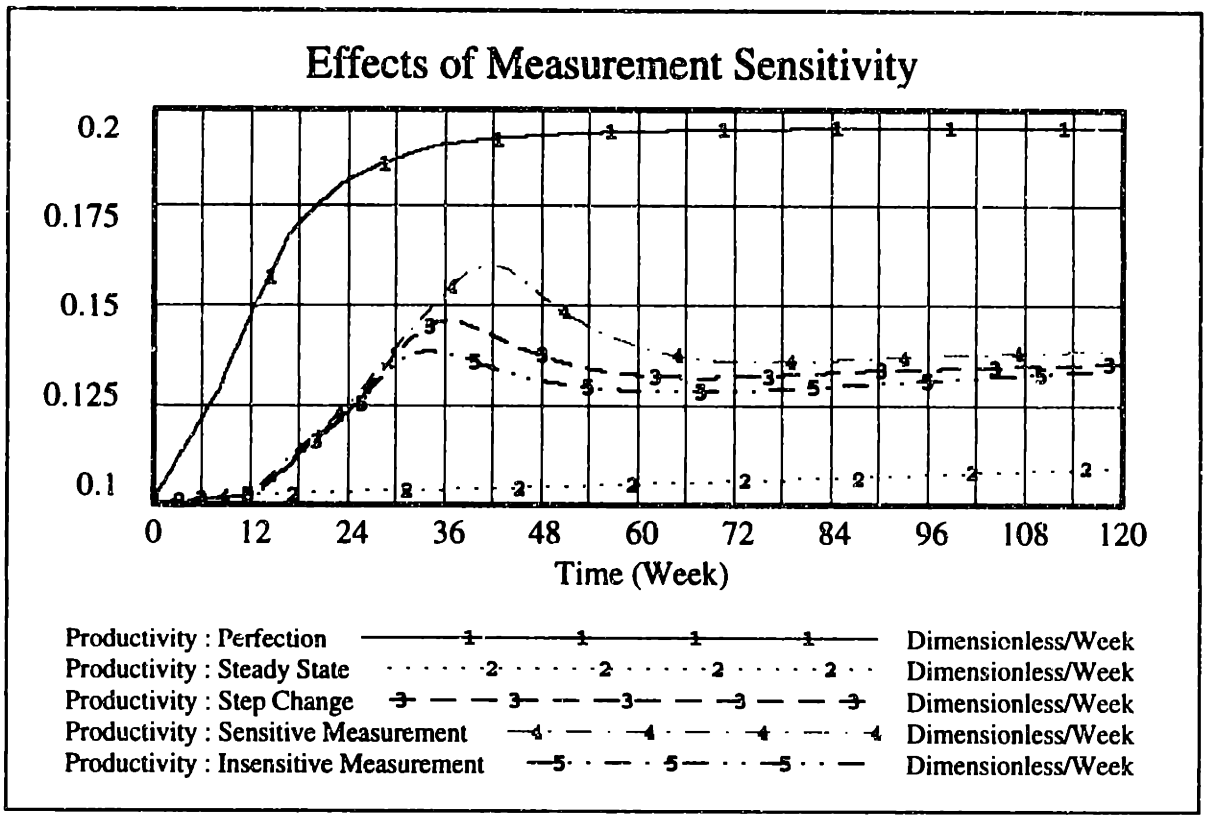

Figure 18, Effects of Measurement Sensitivity

The key factors required to make a lean manufacturing system work are motivation and the improvement process. Of the two, motivation is the most important. If motivation is high, the system jump-starts itself and produces excellent results. Low motivation will prevent the system from functioning at all. The improvement process is key because it provides the means for combining the information provided by the kanban system and the motivation of the workers. The better it functions, the better the results. Although an only moderately productive process of 50%-70% works very well.

Lean manufacturing discussions tend to focus on the information-gathering aspects of the system as represented by the kanban system. That is only one-third of the story. The reinforcing improvement loop doesn't start itself simply because it is put in place. The proper enabling environment, including motivated workers, a good improvement process, low inventory, and good measurement, must also be in place.

VI. Software Project Model:

A. Basic Modeling Decisions

The second model in this investigation was much easier to build than the first. The cell model was built from scratch, where the project model is the cell model with the assembly cell replaced by a standard project model as developed at Pugh-Roberts Assoc [2] [8] [9]. (I built my copy of the basic project model as part of a project management class guided by Jim Lyneis of Pugh-Roberts Associates.)

The tricky part of replacing the cell with the project was understanding how and why to make the connections. The key is to understand the similarity of concepts from lean manufacturing and the software project presented in the case study. Table 5 shows a simplistic mapping between the lean manufacturing ideas and ideas from the case.

Lean Manufacturing	Case Study
Customer Value	Focus on result
Kanban	PRCR, Integration, Configuration Management, Provide feedback quickly
Jidoka	Remove opportunity for error, Standardize, Optimize, Automate
Process Improvement	Peer Pressure, Problem solving expected, Teams, Make right easier than wrong
Standard Work	Standards, Standard Interfaces

Table 5, Mapping of Lean Manufacturing onto Case Study

The mapping starts with the focus on customer value. Although the idea is summarized better in lean manufacturing, "Focus on result" is the same idea. It explains and, in part, justifies the rest of the comparisons made between lean manufacturing and software development as described in the case. Just as picking your axioms defines the rest of the system in geometry, selecting a common goal is very likely to produce similar results.

Standard work to standards and standard interfaces is the easiest connection to make. The standard process sheets described by Ohno correlate directly with documented standards discussed in the case. Standard interfaces are less obvious until you consider the nature of assembly. In order to put parts together, their design must include compatible interfaces. Standard thread sizes for bolts are an example. Standard interfaces for software are exactly the same and provide similar benefits.

The connections between jidoka and the case are also fairly easy to explain. "Remove the opportunity for error" is the jidoka concept. "Standardize, Optimize, Automate" is the embodiment of the process of implementing jidoka. Yes, software is different than manufacturing, but the ideas behind them are identical.

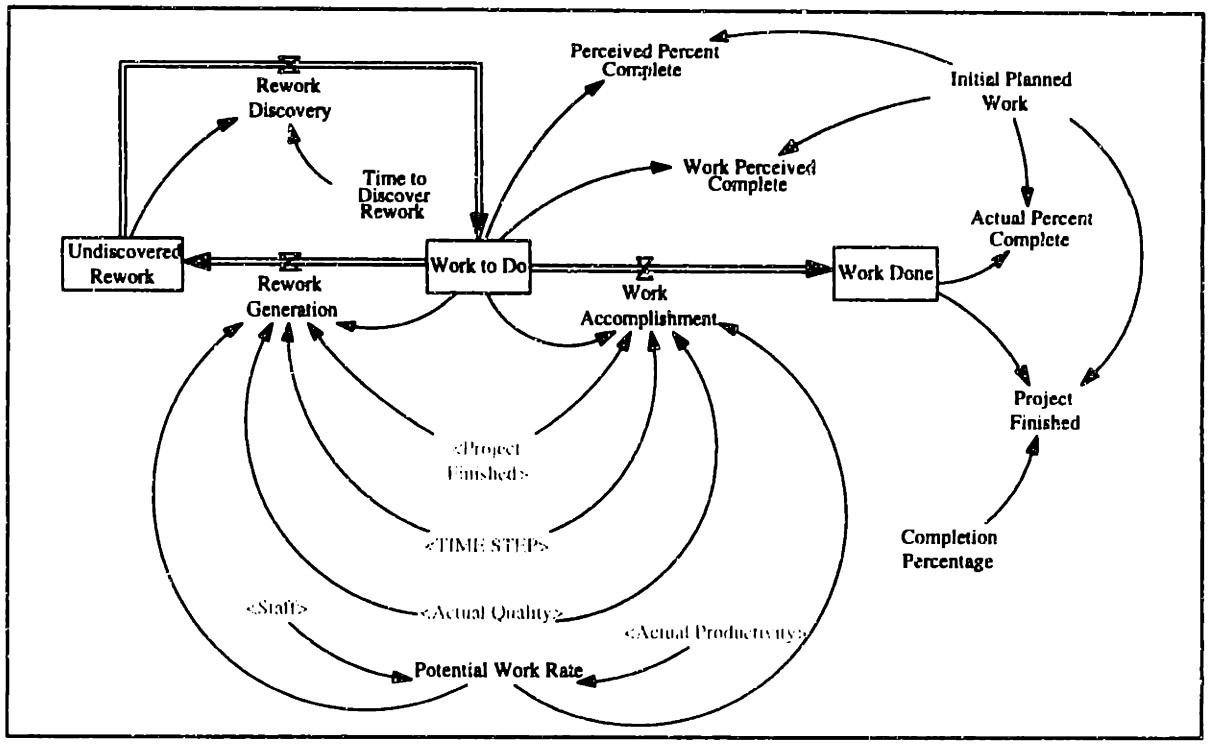

Figure 19, Rework Loop

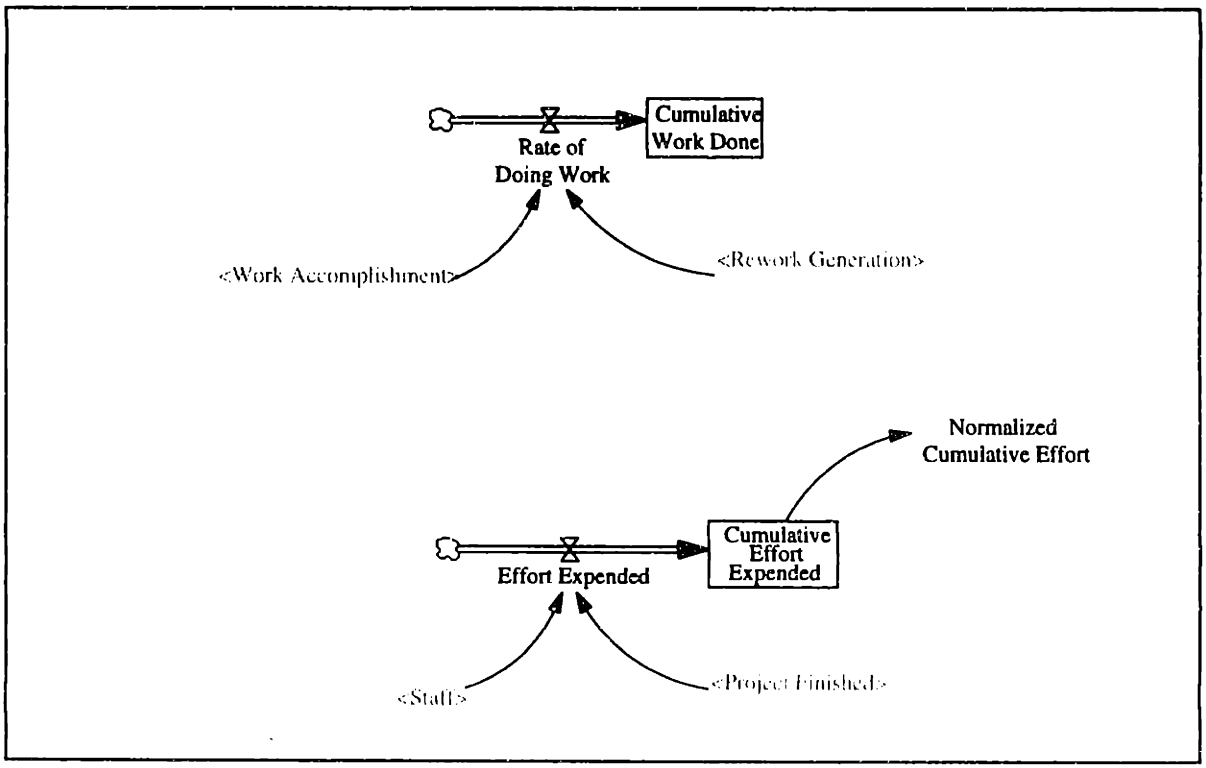

Figure 20, Cumulative Work

Process improvement is similar to jidoka except the case addresses the human element more directly than lean manufacturing. "Make right easier than wrong" provides guidelines on how to produce processes that are inherently easy to follow. Software development has a greater need for this than manufacturing, since the leeway for developers to develop alternative methods is greater. Peer pressure is implemented differently, but the concept of keeping progress visible is constant. (Lean manufacturing uses Andon Boards and the case utilized status meetings.) The expectation of problem solving is also more direct in the case. A large part of the difference is the expectations between engineers and production workers. Production workers tend to need more guidance and encouragement to provide effective problem solving. Engineers are trained in analysis and problem solving, so it is more a matter of focusing their energies on the right problems.

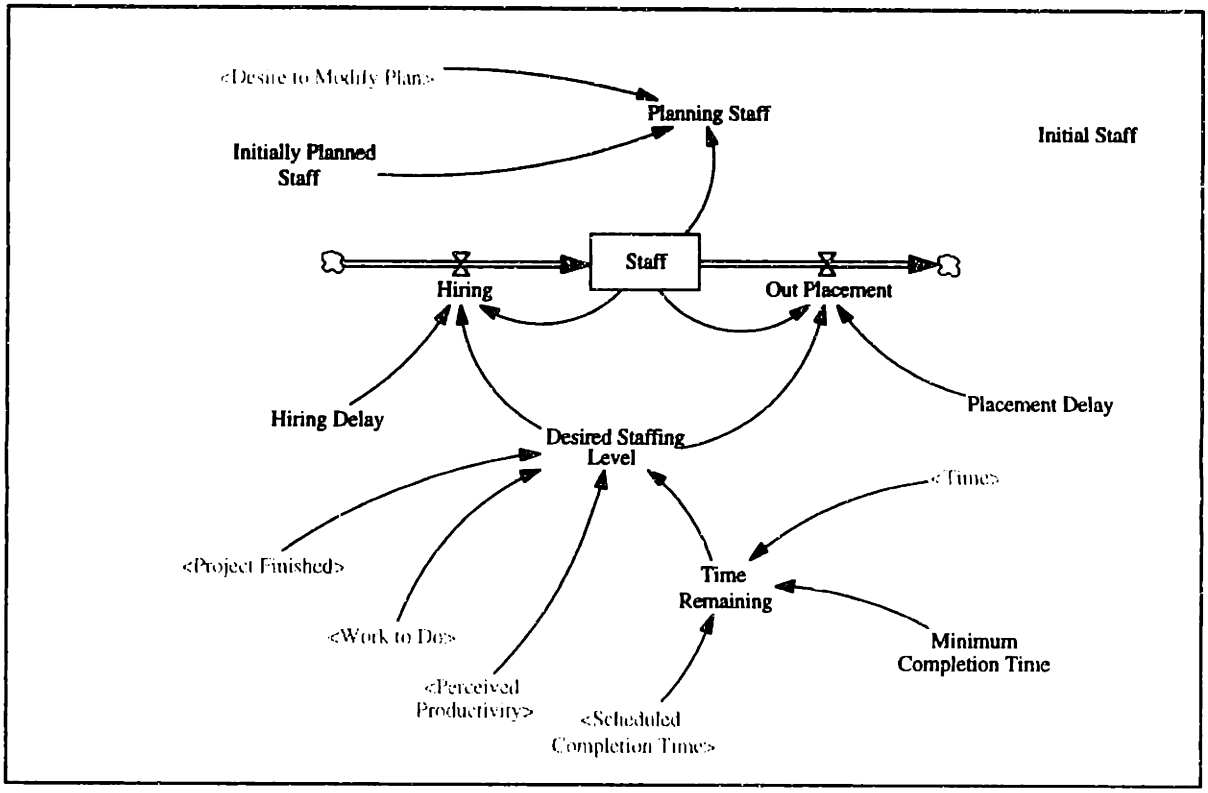

Figure 21, Staff Level

Kanban for manufacturing is a simple and concise way of providing control and feedback as described by Ohno in Table 1. For software development, the desired benefits require a more complicated mosaic. Part of the difference is the time scale. Manufacturing tends to provide rapid feedback naturally. "Provide feedback quickly" has to be built into a software development system explicitly. The integration methods in the case provide an example. Traditional integration slowly provides small amounts of feedback. The techniques used in the case had to be implemented specifically to prevent the natural desire to delay integration. Configuration management, coupled with the PRCR system, provides the basic kanban functions of control, tractability and accountability.

B. Rework Loop

The rework loop consists of three stocks. *Work to Do*, *Work Done*, and *Undiscovered Rework*. It is the classic rework cycle as described in the literature [2]. In this formulation, the total amount of work is constant. No additional work is added after the start of the project. *Project Finished* is a special variable used to indicate when the project has been completed.

C. Cumulative Work

Cumulative Work Done and *Cumulative Effort Expended* are, along with *Project Finished*, the main measures of the model. *Cumulative Work Done* is the total amount, including rework, of work accomplished. Unless quality is 100%, it is always greater than Total Initial Work. *Cumulative Effort Expended* is the total person-months spent on the project. It is the integral of the *Staff* on the project.

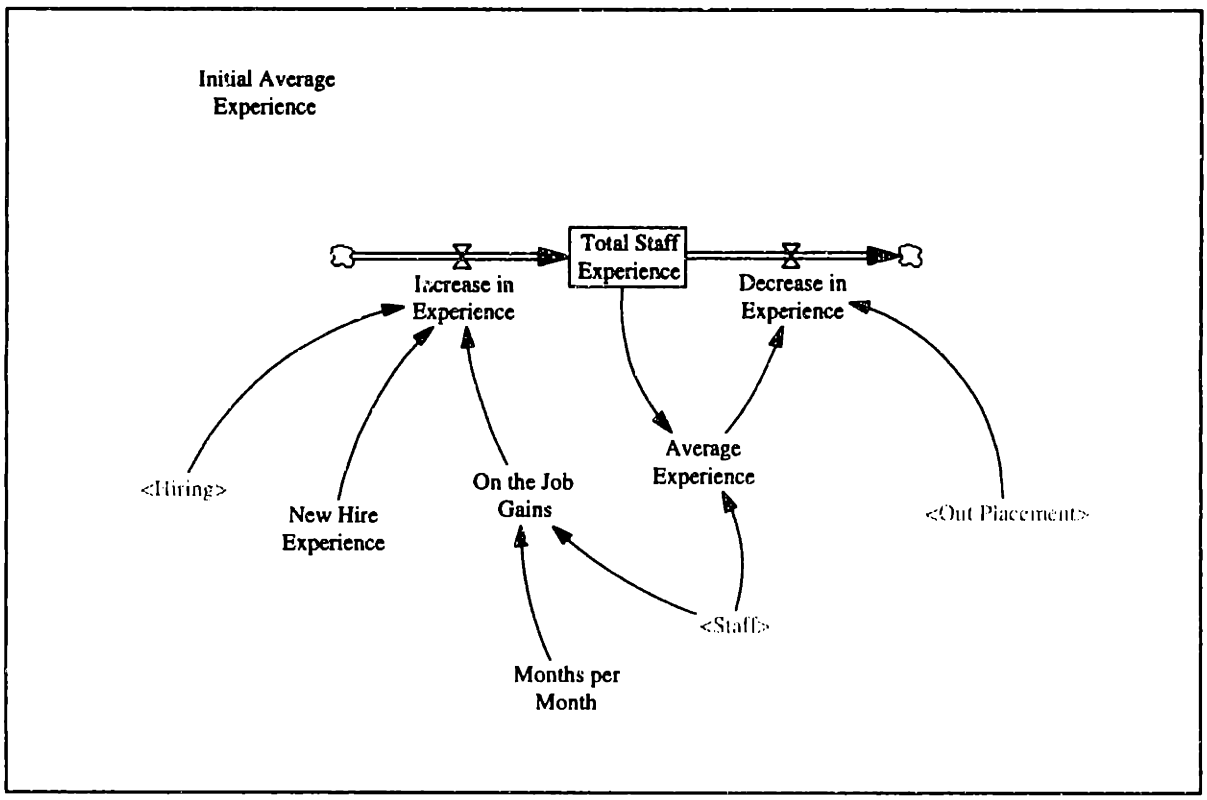

Figure 22, Staff Experience

D. Staff Level

Staff is the number of people on the project. After being initialized by the *Initial Staff*, *Staff* is adjusted to meet the *Desired Staffing Level* within the constraints of the *Hiring Delay* and the *Placement Delay*. *Desired Staffing Level* is calculated in a simple linear fashion based on the remaining work, time, and measured productivity.

Planning Staff is the number of people assumed for planning purposes. At the beginning of the project it is held constant at the planned staff count for the project. As the project progresses and the *Desire to Modify Plan* increases, it eventually becomes equal to *Staff*.

Initial Staff plays in important role in the dynamics of the project model. I have selected initial staff to be slightly understaffed because that is the normal condition of a software project. It also prevents the effects of severe under-staffing and over-staffing from coloring the results of the analysis.

E. Staff Experience

This portion of the model is selected to model the effects of on-project experience. It uses a standard coflow structure to determine the overall staff experience as a function of *New Hire Experience*, *Hiring*, and *Out Placement*. An alternative formulation would have been to model overall, on and off project, experience of the *Staff*. The on-project formulation was chosen because it more closely represents the case study.

Software development in general varies greatly from company to company and project to project. Technology is marching so quickly that twenty year veterans and recent graduates face the same learning curves at the start. (Some even argue that less experience is better because recent graduates tend to be schooled in the latest ideas and do not have to unlearn bad habits.) On average, the ancient ones' familiarity with the problem domain and the young ones' clean slate balance out. Thus time on the project becomes the dominant factor in productivity and quality.

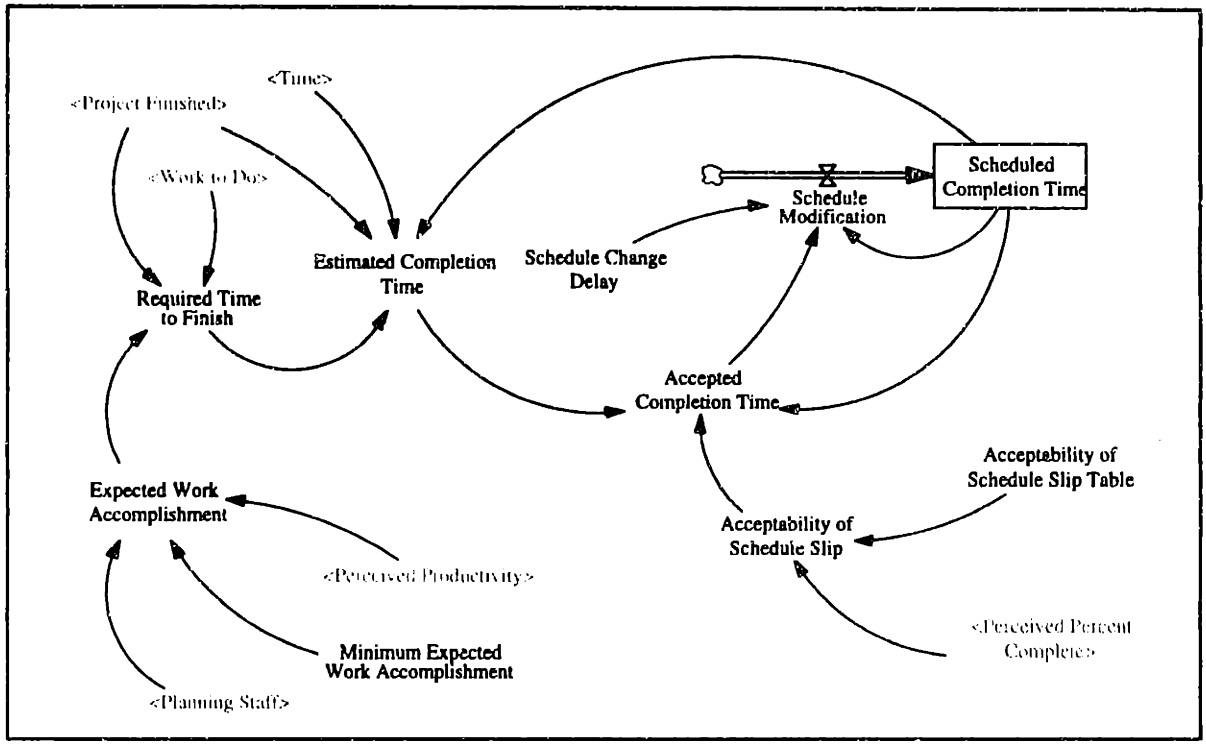

Figure 23, Schedule

F. *Schedule*

Although it looks complicated, the schedule section of the model is conceptually fairly simple. The basic function is to calculate *Estimated Completion Time* from the remaining work, staff, and measured productivity. Actual changes to the schedule are made based on the *Schedule Change Delay* and the *Accepted Completion Time*. The tricky part is modeling the transition from the initial plan to a new plan based on the state of the project.

At the beginning of a project, the startup effects are considered and planners generally stick to the original staffing and schedule until they feel that the project is running as desired in the middle portion of the project. Once in the middle of the project, schedule changes are generally accepted based on the *Estimated Completion Time*. Then, at the end of the project, the desire to change the schedule is reduced so that only short-term schedule slips are allowed. *Acceptability of Schedule Slip* models these effects based on *Perceived Percent Complete*.

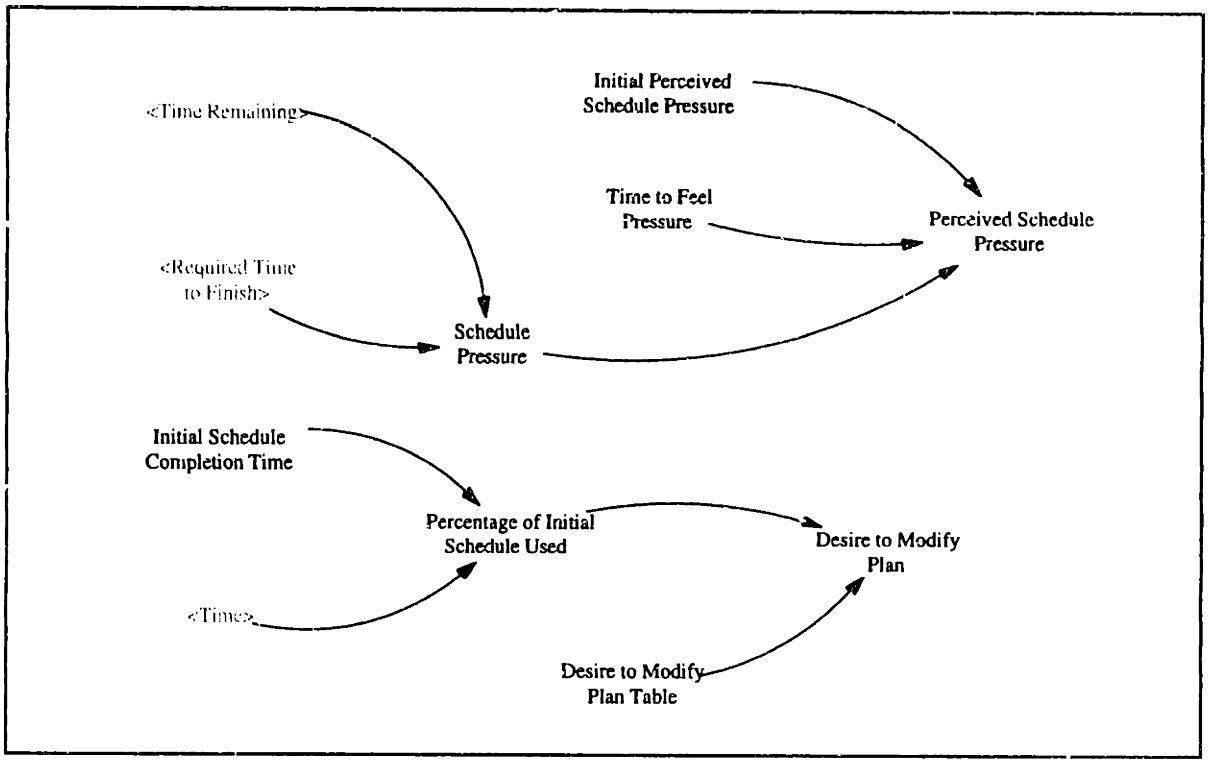

Figure 24, Schedule Pressure and Plan Modification

G. Schedule Pressure

This section of the model has two parts. The first, *Desire to Modify Plan*, is used to model the transition from using the *Initial Planning Staff* to *Staff* for planning purposes. It is similar to *Acceptability of Schedule Slip* in the previous section. The only difference is the continued desire to plan using the current staff at the end of a project. It captures the typical willingness to throw as many people as required at a project to complete it as the end approaches.

The second, *Perceived Schedule Pressure*, is critical to the rest of the model. It models the response of the work-force to the differences between the project plan and its actual state. Those differences, with an appropriate delay, drive quality and productivity effects discussed later.

H. Productivity

Productivity has two facets in this model. There is *Actual Productivity* and *Perceived Productivity*. Like the name indicates, *Actual Productivity* is the real productivity of the project at each moment in time. *Perceived Productivity* is the measurable productivity for the project. The difference is driven from the unknowable nature of undiscovered rework. Since it is undiscovered, project planners do not know what it is and cannot take it into consideration. (Quality measures could be used to estimate it, but are seldom used.)

Actual Productivity consists of the sum of the basic veteran productivity modified by the effects of schedule pressure and experience. Unless the entire team has achieved veteran status, the *Experience Ratio* will be less than one and *Actual Productivity* will be lower than the expected

veteran productivity. As schedule pressure mounts, productivity increases as the staff works harder to catch up. After a certain point, people are working as hard as possible, and increases in schedule pressure have no effect. (This model doesn't include burnout factors such as excessive overtime and prolonged schedule pressure.)

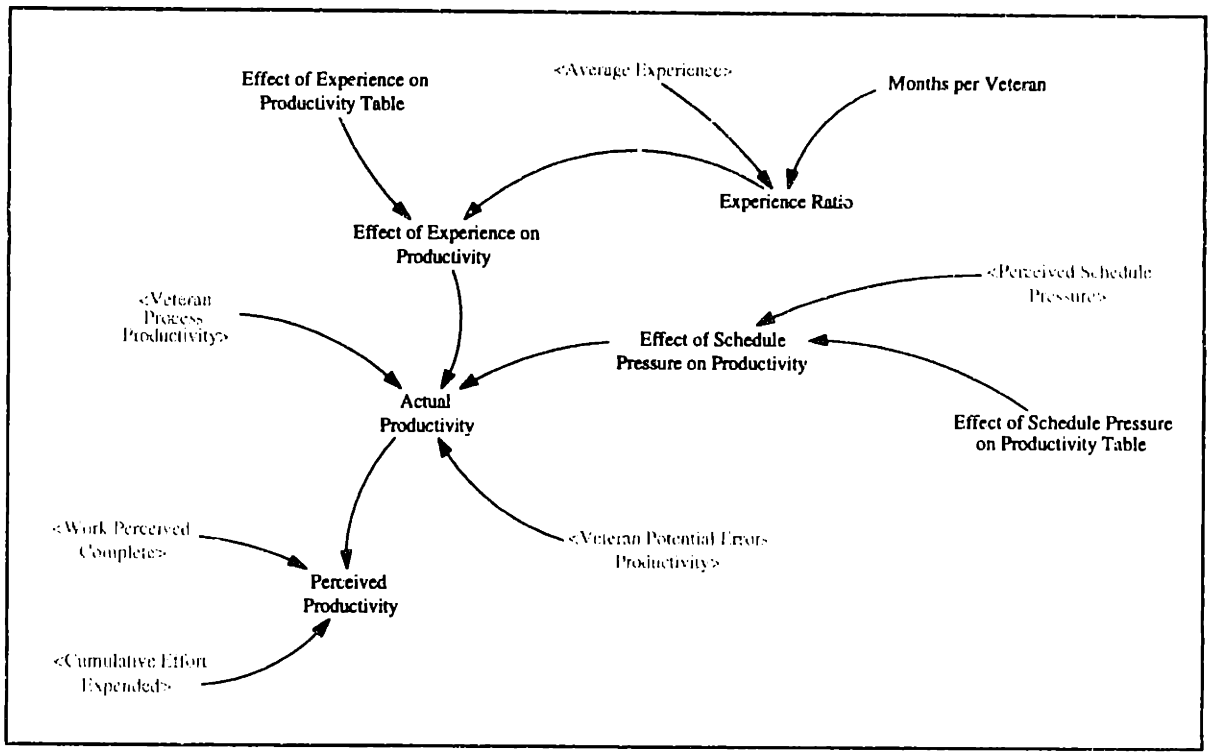

Figure 25, Productivity

In the basic Pugh-Roberts model, veteran productivity is a constant. This model splits the constant between potential errors and process. I chose to assign 70% of the productivity to the process and 30% to potential errors. The decision is somewhat arbitrary based on the assumption that the majority of productivity is based on process but probably no more than 75%. The base productivity increases as potential errors are reduced and process improves to a maximum of forty (12 for errors and 28 for process). The range from initial to possible best productivity was selected to match the range of the cell model.

I. Quality

Quality is modeled very similarly to productivity. The basic value of quality is treated identically to that of productivity, with a maximum value of one (0.7 for process and 0.3 for errors). The effects of schedule pressure and experience are modeled with the addition of the effect of quality on quality. High schedule pressure tends to diminish quality. As people start working faster to increase productivity, the quality of their work falls. Just like productivity, only an experience ratio of one allows the project to experience the benefits of veteran quality.

The effect of quality on quality is more subtle. It is based on the idea that current work tends to be built on completed work. As long as the completed work was done correctly the normal quality factor for the new work applies. However, when the completed work is incorrect, the new

work on which it is based will also be wrong, regardless of the current quality. In other words, good work based on bad work is also bad.

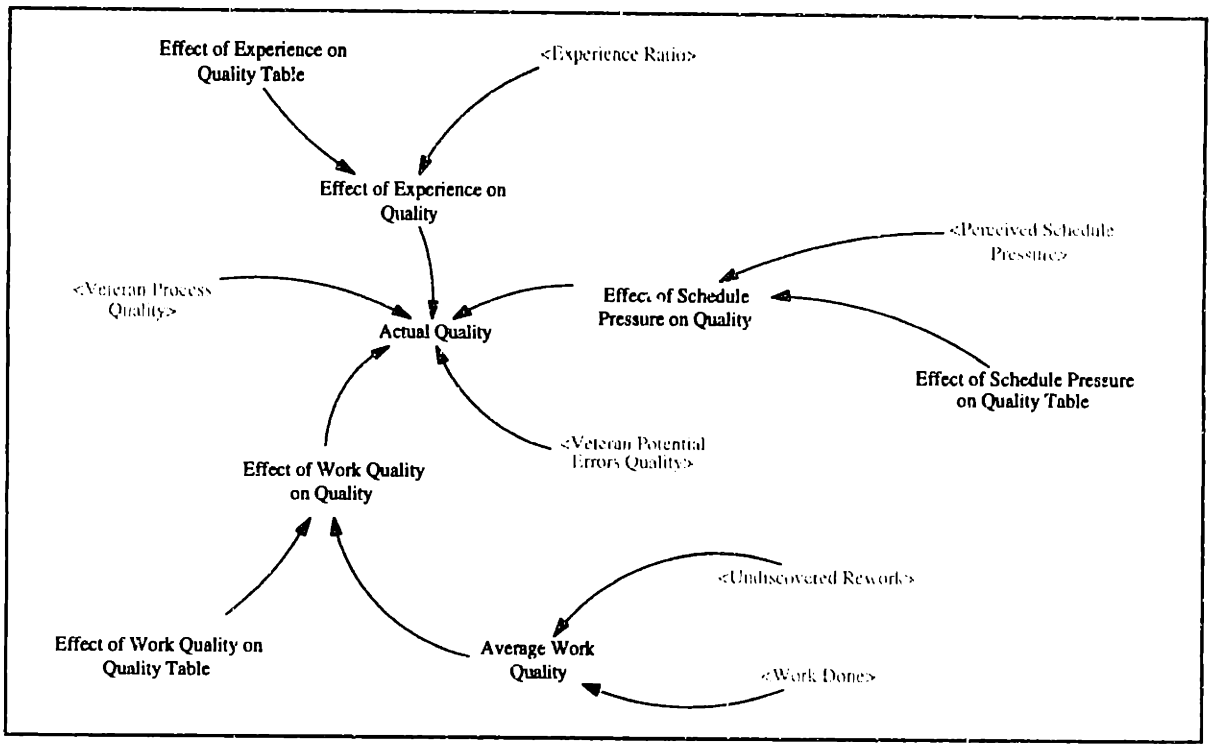

Figure 26, Quality

J. Jidoka (Potential Error Removal)

This section of the project model is identical, as I promised, to the cell model except for the translation of *Percent Potential Errors* into quality and productivity. In the cell model I aggregated the quality and productivity concepts into a simple delay. The project model formulation prevents that basic simplification requiring separate representations of both quantities. The lookup table for the delay was inverted and scaled for the different quantities. Both quality and productivity are scaled to represent 30% of the total. (A future refinement to the model would be to normalize the tables and use input parameters to scale the maximum values and control the distribution.)

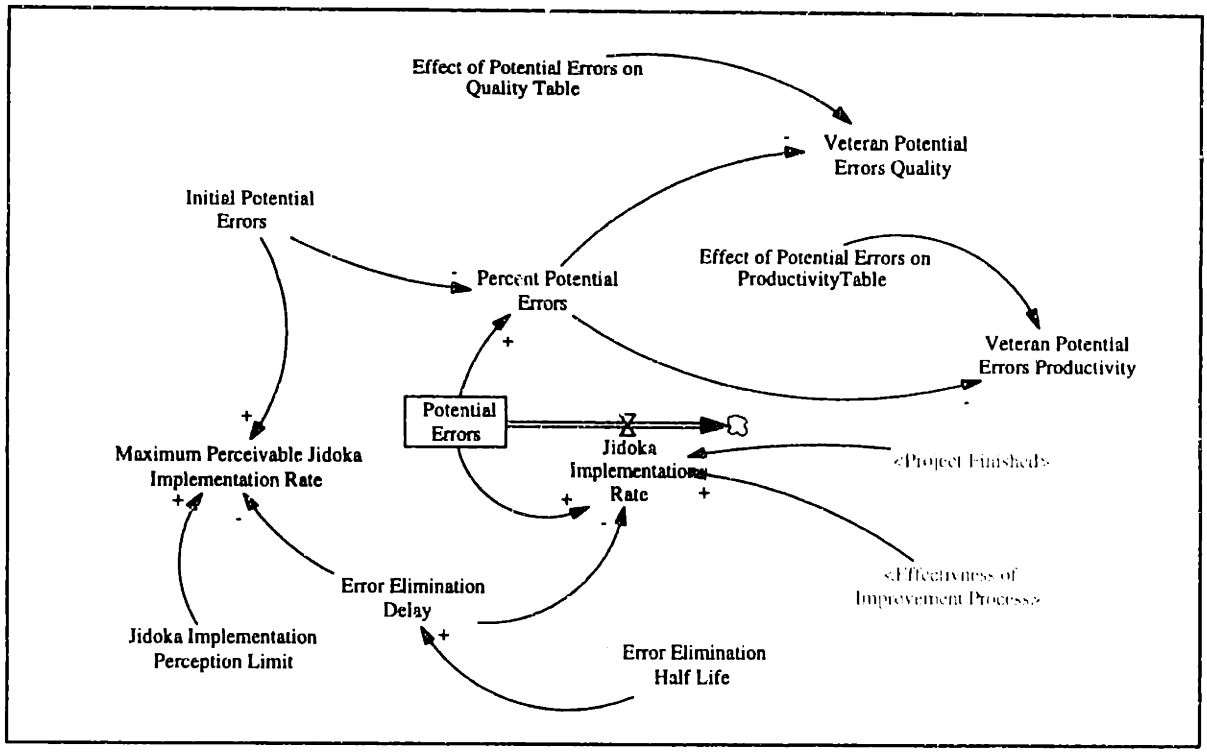

Figure 27, Jidoka (Potential Error Removal)

K. Process

This section is identical to the improvement process section of the cell model.

L. Process Quality and Productivity

The process quality and productivity portions of the project model are structurally identical to the process work completion delay section of the cell model. They are simply broken out with the 70/30 rule applied to the nominal and best values. The diagrams flow in the opposite direction because quality and productivity are the reciprocal of the delay. Thus entropy decreases quality and productivity in the project model where it increases the delay in the cell model. Project Finished was also added to the rates to turn them off when the project was completed.

M. Motivation

This section is also structurally identical to the cell model. One minor change is to break the process motivational factor into productivity and quality motivational factors. The second change is to substitute *Perceived Schedule Pressure* for *Production Stress*. Kanban systems supply their feedback from the movement of work in the cell. The equivalent feedback from the case is in schedule pressure. Problems with integration and excess PRCRs result in schedule pressure, which is transmitted to the workers at the weekly status meetings.

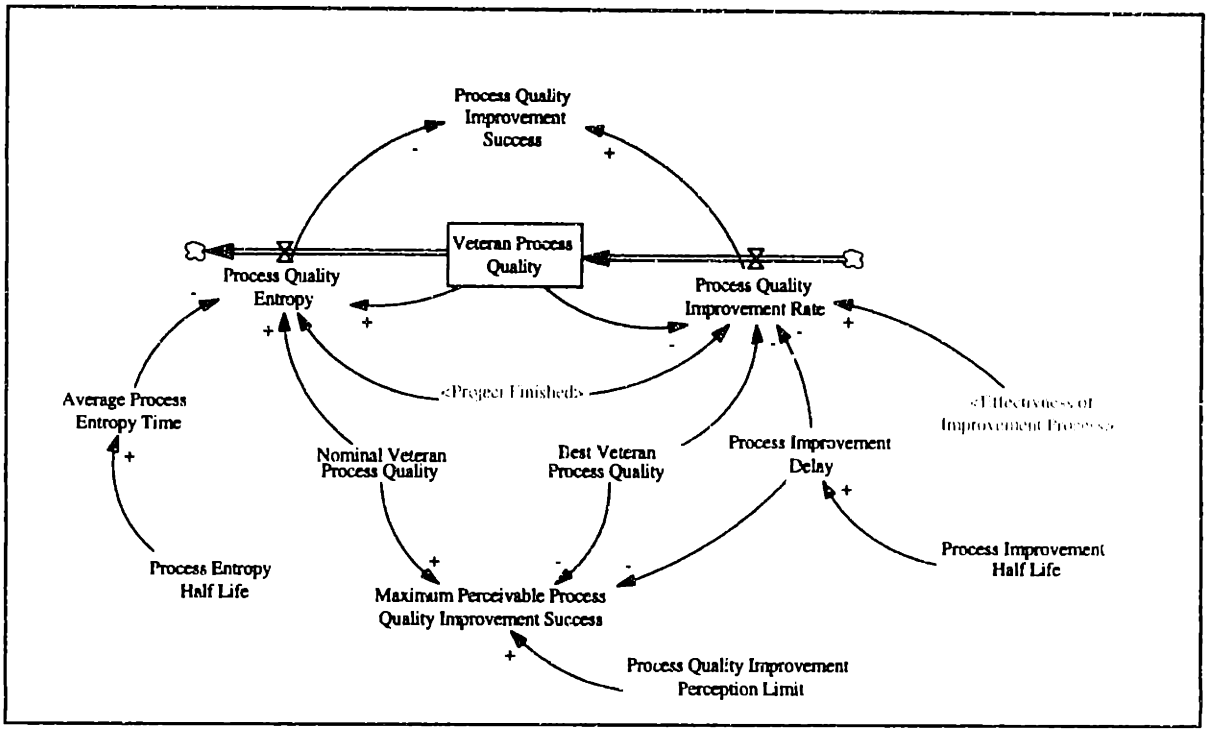

Figure 28, Process Quality

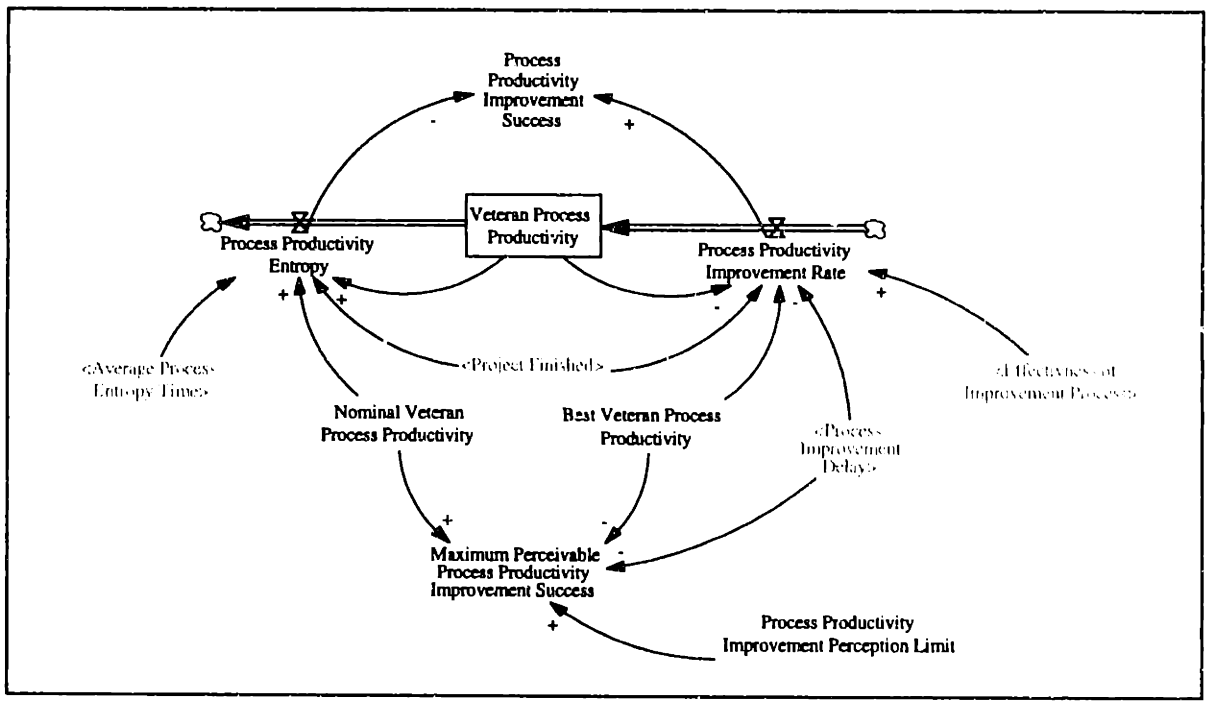

Figure 29, Process Productivity

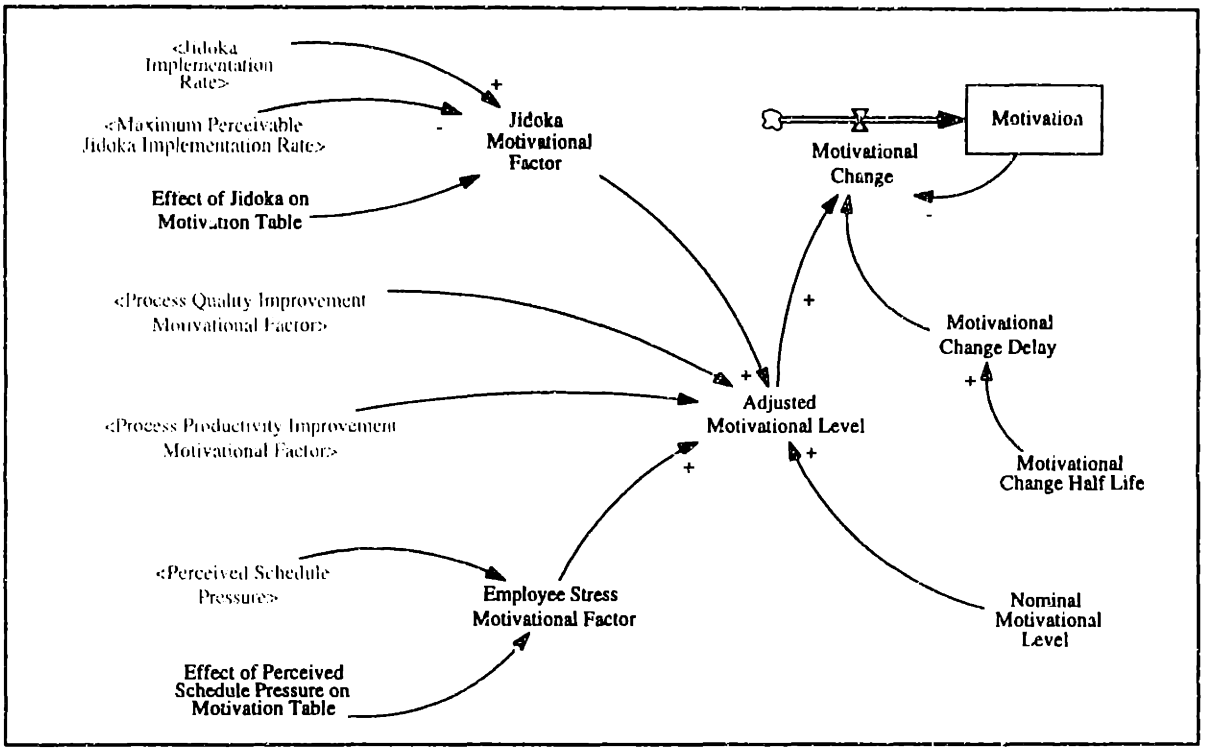

Figure 30, Motivation

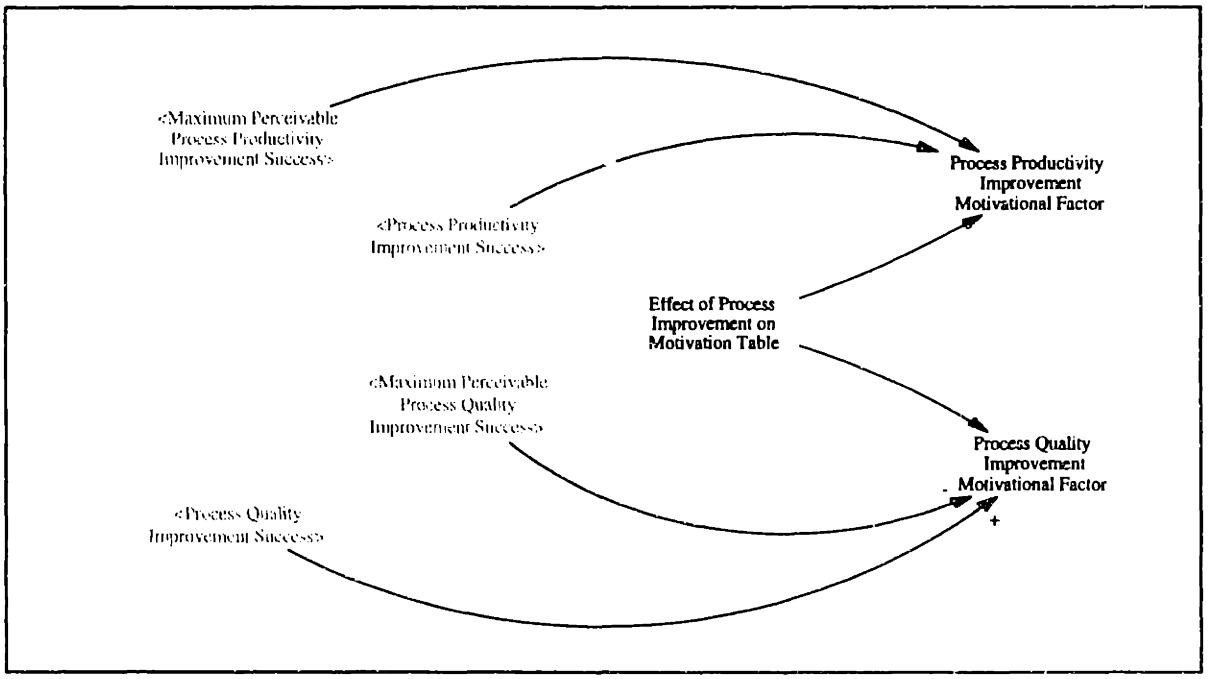

Figure 31, Motivation (Cont. ...)

VII. Software Project Model Analysis:

Project models are complicated and provide a great deal of opportunity for analysis. This investigation is limited to the effects of the lean development loops borrowed from the cell model. The project model was run using the same parameter settings used in the cell model analysis. Since neither concept applies to a development project, the "step change" and "high inventory" cases were omitted. The "reference base" case is the project with the lean development loops turned off.

	Cumulative Effort	Project Duration	Cumulative Work	Cumulative Effort	Project Duration	Cumulative Work
Perfection	259.94	50.25	4,117	47%	82%	87%
Success	279.22	51.19	4,199	50%	83%	89%
Good Process	307.30	51.56	4,324	55%	84%	91%
High Motivation	346.39	55.06	4,391	62%	89%	93%
Sensitive Measurement	375.15	55.50	4,483	67%	90%	95%
Steady State	378.55	55.56	4,490	68%	90%	95%
Insensitive Measurement	379.57	55.56	4,492	68%	90%	95%
Low Motivation	447.83	56.81	4,599	80%	92%	97%
Bad Process	474.71	60.19	4,653	85%	98%	98%
Failure	558.72	61.63	4,736	100%	100%	100%
Reference Base	2622.00	92.25	5,049	469%	150%	107%

Table 6, Project Model Summary Data

For a project model, Cumulative Effort, Project Duration, and Cumulative Work summarize the success of the project. Table 6 shows the data collected from the various cases. Figure 32 graphs the normalized data, excluding the "reference base" case. It is excluded so that the variation due to the model parameters is easier to see.

The results have several interesting characteristics. Most surprising is the huge difference between the "reference base" case and all the others. Even the "failed" case is vastly superior. (Especially in an effort which is four-and-one-half times larger.) In part, this is due to the leverage that productivity and quality have in a project model. It is also due to my choice of constants and final values for productivity. Without gathering a great deal of data, there is no way to accurately calibrate the model.

Also surprising to me were the effects of *Nominal Motivational Level* and *Quality of Improvement Process*. In the cell model, motivation had the biggest impact, followed by process quality. For the project model the order was reversed. The difference is due to the strength of the motivational loops in the project model. Even with low levels of motivation, they quickly reach and maintain their maximum level.

The most important result is that the loops function as expected. The lean manufacturing constructs function very well on a project. Clearly implementing, even badly, the lean software ideas from the case results in dramatic improvements. A closer look at how the model works explains the dramatic improvement.

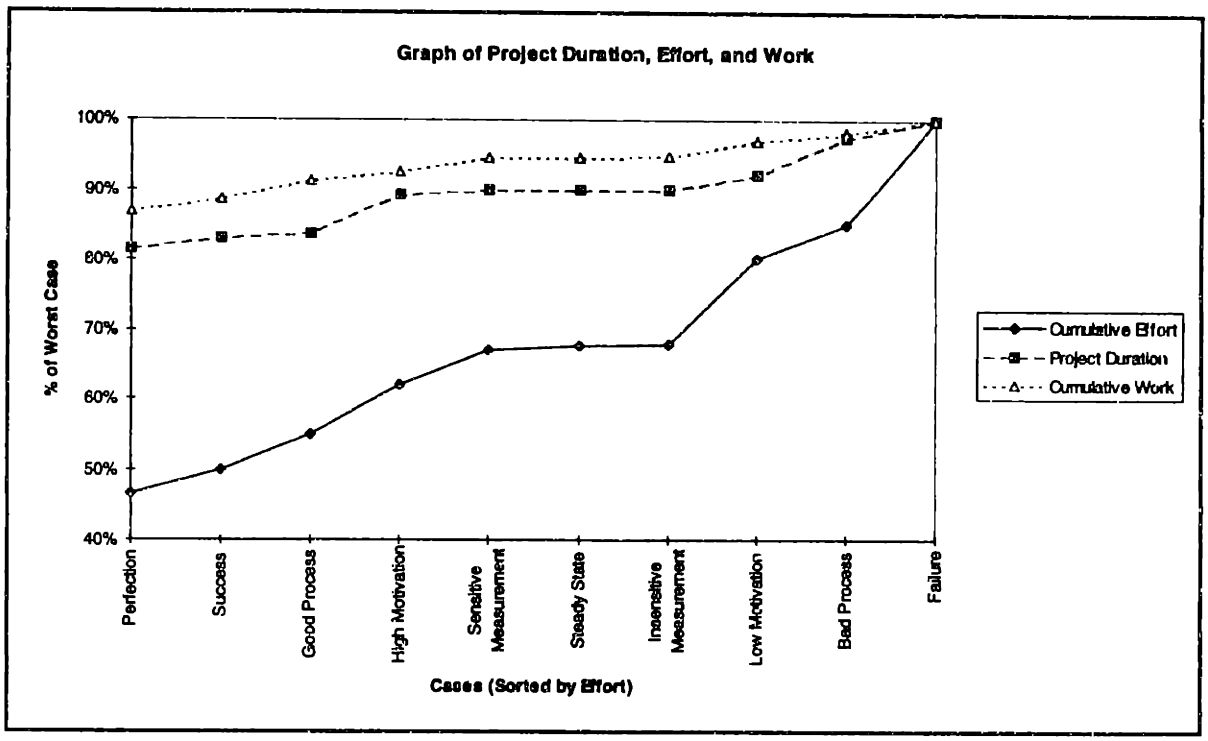

Figure 32, Project Duration, Effort, and Work

The bulk of the original project model is structure that produces negative effects on productivity and quality. Starting the project requires hiring staff, which dilutes the average experience, thereby reducing quality and productivity. Schedule pressure increases productivity but reduces quality, so that the same work is done wrong more often. This increases total effort. The effect of quality on quality reduces the overall quality by definition. With all of the reductions in productivity, schedule pressure is driven to very high levels.

The resulting state of the project is ideal for the lean software improvement loops. The casual loop diagram in Figure 11, replacing production stress with schedule pressure, still applies. The naturally occurring schedule pressure in the project drives the improvement loop very strongly. It supplies enough motivation that the *Quality of Improvement Process* becomes more important than the *Nominal Motivational Level* in determining the rate of improvement. The strong improvement loop drives up productivity overpowering the early negative effects from the rest of model. By the time the reductions in pressure turn off the improvement loop, the project is over, so there is no opportunity for entropy to reduce productivity.

Although not specifically covered in the case study, the results from the model mirror my own experience. New projects often start with bursts of improvement with or without any formal development or improvement processes. It makes sense that creating an outlet to channel those natural tendencies would produce dramatic improvements.

VIII. Conclusion:

The objective of this investigation was to determine if the principles of lean manufacturing were applicable to product development, and more specifically, software development. Proving that lean software development exists and can be practiced consisted of three tasks.

- Develop a model of a lean assembly cell based on lean manufacturing principles.
- Present an actual case of software development where lean manufacturing ideas had been applied.
- Based on the case study, add structure developed in the assembly cell model and demonstrate behavior similar to the case.

The cell model clearly demonstrates the structures and behaviors of a lean manufacturing assembly cell. By properly selecting the input parameters, the model will produce behavior modes consistent with the body of experience with lean manufacturing. The cell model demonstrates how a lean manufacturing system works and the importance of worker motivation and improvement process quality.

The case study presents real-world data about a very successful software development project. The philosophy and principles used to guide the project are identical to those of lean manufacturing. The implemented processes closely mirror lean manufacturing, as well.

The project model is a standard project model modified with the improvement structures using the case study as justification. The modified model produces improvements in cumulative effort and schedule duration consistent with the case study. It also exhibits similar sensitivity to motivation and improvement process quality.

The structures are the same. The principles from which they are derived are the same. The results are similar and they correspond to real world data. Therefore, lean manufacturing ideas must be applicable to software development. Although the specific implementations differ, the fundamental similarities cannot be ignored. While not well-developed or practiced, the concept of lean software development is as valid as lean manufacturing.

Actually, I am convinced that Womack and Jones are right in asserting the principles of "Lean Thinking". Given any system or endeavor, the proper application of Ohno's principles, starting with customer value, should yield desirable results. The art is seeing and understanding how to create policies and structures that implement the principles.

In addition to the overall objective of my investigation, I have learned several valuable lessons about lean manufacturing and software development, as well. Most important is the role of jidoka style improvements. By removing the human element, they cannot succumb to entropy. This makes them the most valuable form of method.

The role of adjusting the number of staff to match the demand rate was a surprise. In order to maintain the improvements in process, the high level of performance must always be required. If a task is over-staffed, there is no need to perform at a high level, and entropy takes over until the available level of performance matches demand. Expectations truly drive the results.

Implementing lean anything is a difficult task. In lean manufacturing, the structure is well documented, but motivation - which is very difficult to manage or even measure - is just as important. In lean software development the difficulty is compounded by the complexity of the structure. In order to produce the same effects as the kanban system, a complex web of PRCRs, testing, integration, and staff meetings must be constructed.

The last big learning is an increased appreciation for the value of system dynamics as a tool. At Toyota it took many years of experimentation to develop the right implementation. If system dynamics had been available to Taiichi Ohno, he could have retired at a much younger age. Having been involved in developing the software development processes described in the case study, looking at them from a system dynamics perspective gives me much greater insight. I would never again undertake such a project without building a model.

IX. Further Investigation

With any investigation, answering one question often leads to two more. This one is no exception. Below is a list of areas ideas for modifying and extending this current work.

1. Extend models to include motivation and quality of improvement process.
2. Refine cell models to explore the nuances of the kanban system.
3. Extend the model to reflect a multi-cell production line.
4. Extend project model to include a cascaded series of projects.
5. Extend project model to include aspects of the case data to explore motivation and the quality improvement process in more detail.
6. Include market forces and technological change in project model.

X. References

1. Cochran, David S. (1994), The Design and Control of Manufacturing Systems, PHD Dissertation Auburn University 1994
2. Cooper, Kenneth G., THE $2,000 HOUR: How Managers Influence Project Performance Through the Rework Cycle Project Management Journal, vol. XXV, No.1, March 1994
3. Cusumano, Michael A. And Richard W. Selby (1995), Microsoft Secrets, The Free Press
4. Forrester, Jay W. (1961), Industrial Dynamics. Portland OR: Productivity Press.
5. Forrester, Jay W. (1991), System dynamics and the Lessons of 35 Years.
6. Hayes, Robert H. (1981) Why Japanese Factories Work, Harvard Business Review, July-August 1981
7. Kaplan, R. (1990a) Analog Devices: The Half-Life System, Case 9-191-061, Harvard Business School.
8. Lyneis, James M. (1980), Corporate Planning and Policy Design. Portland OR: Productivity Press.
9. Lyneis, James M. (1996), The Calibration of System Dynamics Models, Pugh-Roberts Associates
10. Monden (1993), Toyota Production System, ISBN 0-89806-129-6
11. Nahmias, Steven (1997), Production and Operations Analysis 3rd Edition 6.6 and 6.7, Times Mirror Higher Education Group
12. Ohno, Taiichi (1988), Toyota Production System: Beyond Large-Scale Production, Productivity Press
13. Shingo, Shigeo (1981), Study of 'Toyota' production system from industrial engineering viewpoint, Tokyo : Japan Management Association
14. Sterman, John; Repenning, Nelson; and Kofman, Fred (1994); Unanticipated Side Effects of Successful Quality Programs: Exploring a Paradox of Organizational Improvement; Management Science vol. 43, No. 4, April 1997
15. Sterman, John D. (1994), Learning in and about complex systems, System Dynamics Review vol. 10, nos. 2-3 Summer-Fall 1994
16. Toyota Motor Corporation (1992), *The Toyota Production System*, Operations Management Consulting Division
17. Womack, James P.; Jones, Daniel T.; and Roos, Daniel (1990); The Machine That Changed The World, New York: Rawson Associates
18. Womack, James P.; Jones, Daniel T.; (1996); Lean Thinking, New York: Rawson Associates

XI. Appendix A, Cell Model

A. Command File

This is the Vensim command file used to generate the data sets used in the analysis of the cell model.

```
SPECIAL>LOADMODEL|cell.mdl
SIMULATE>READCIN

SIMULATE>RUNNAME|Steady State
MENU>RUN|o

SIMULATE>SETVAL|Step Height = 0.5
SIMULATE>RUNNAME|Step Change
MENU>RUN|o

SIMULATE>SETVAL|Step Height = 0.5
SIMULATE>SETVAL|Nominal Motivational Level = 0.4
SIMULATE>RUNNAME|Low Motivation
MENU>RUN|o

SIMULATE>SETVAL|Step Height = 0.5
SIMULATE>SETVAL|Nominal Motivational Level = 0.6
SIMULATE>RUNNAME|High Motivation
MENU>RUN|o

SIMULATE>SETVAL|Step Height = 0.5
SIMULATE>SETVAL|Quality of Improvement Process = 0.4
SIMULATE>RUNNAME|Bad Process
MENU>RUN|o

SIMULATE>SETVAL|Step Height = 0.5
SIMULATE>SETVAL|Quality of Improvement Process = 0.9
SIMULATE>RUNNAME|Good Process
MENU>RUN|o

SIMULATE>SETVAL|Step Height = 0.5
SIMULATE>SETVAL|Total Initial Work = 30
SIMULATE>RUNNAME|High Inventory
MENU>RUN|o

SIMULATE>SETVAL|Step Height = 0.5
SIMULATE>SETVAL|Process Improvement Perception Limit = 0.4
SIMULATE>SETVAL|Jidoka Implementation Perception Limit = 0.4
SIMULATE>RUNNAME|Sensitive Measurement
MENU>RUN|o

SIMULATE>SETVAL|Step Height = 0.5
SIMULATE>SETVAL|Process Improvement Perception Limit = 1.0
SIMULATE>SETVAL|Jidoka Implementation Perception Limit = 1.0
SIMULATE>RUNNAME|Insensitive Measurement
```

```
MENU>RUN|o

SIMULATE>SETVAL|Step Height = 0.5
SIMULATE>SETVAL|Nominal Motivational Level = 0.4
SIMULATE>SETVAL|Quality of Improvement Process = 0.4
SIMULATE>SETVAL|Process Improvement Perception Limit = 1.0
SIMULATE>SETVAL|Jidoka Implementation Perception Limit = 1.0
SIMULATE>RUNNAME|Failure
MENU>RUN|o

SIMULATE>SETVAL|Step Height = 0.5
SIMULATE>SETVAL|Nominal Motivational Level = 0.6
SIMULATE>SETVAL|Quality of Improvement Process = 0.9
SIMULATE>SETVAL|Process Improvement Perception Limit = 0.5
SIMULATE>SETVAL|Jidoka Implementation Perception Limit = 0.5
SIMULATE>RUNNAME|Success
MENU>RUN|o

SIMULATE>SETVAL|Step Height = 0.5
SIMULATE>SETVAL|Nominal Motivational Level = 1.0
SIMULATE>SETVAL|Quality of Improvement Process = 1.0
SIMULATE>RUNNAME|Perfection
MENU>RUN|o
```

B. Model Documentation

These are the equations used in the cell model.

```
Accepted Stress = SMOOTH3 ( Cell Production Stress , Stress Acceptance Delay
        )
        Units: Dimensionless
        Accepted Stress is the level of stress that is accepted as
                normal.

Adjusted Motivational Level = Employee Stress Motivational Factor * Jidoka Motivational Factor
 * Process Improvement Motivational Factor * Nominal Motivational Level
Units: Dimensionless
Adjusted Motivational Level is the Nominal Motivational Level of
                the workers after adjusting for stress and improvement rates.

Average Process Entropy Time = Process Entropy Half Life / 0.7
Units: Week
Average Entropy Time is the time conversion of the half-life.

Cell Production Stress = Effect of Percent Requested Work on Cell Production Stress Table
( Percent Requested Work )
Units: Dimensionless
Cell Production Stress is stress produced by a backlog of
                Requested Work.
```

Change in Pink Noise = (White Noise - Pink Noise) / Noise Correlation Time
Units: Dimensionless/Week
Change in the pink noise value; Pink noise is a first order exponential smoothing delay of the white noise input.

Completed Work = INTEG(Work Completion Rate - Work Removal Rate , Total Initial Work - Maximum Work in Progress - (Work Demand Rate * TIME STEP))
Units: Parts
Completed Parts is the number of parts that have been completed and are waiting for test. It consists of completed parts and their associated kanban cards.

Constrained Desired Average Work Delay = Max (Desired Average Work Delay / Requested Work Adjustment , Minimum Possible Average Work Completion Delay)
Units: Weeks
Constrained Average Work Completion Delay is the Desired Average Work Delay increased by Requested Work Adjustment to and limited by the Minimum delay possible with maximum staffing. (The delay represents both scrap and rework, small delay --> high productivity)

Desired Average Work Delay = SMOOTH ((Maximum Work in Progress / Work Demand Rate) , Desired Average Work Delay Adjustment Time)
Units: Weeks
Desired Average Work Delay is the delay adjusted to meet the demand rate with maximum work in progress. It is calculated so that the Work Completion Rate will match the Work Demand Rate. It is typically adjusted using staff reassignment. (The delay represents both scrap and rework, small delay --> high productivity)

Desired Average Work Delay Adjustment Time = 1
Units: Weeks
Desired Average Work Delay Adjustment Time is the time it takes to adjust the staffing based on improved productivity.

Desired Requested Work = Work Demand Rate * TIME STEP
Units: Parts
Desired Requested Work is the amount of work required to satisfy the current Work Demand Rate. TIME STEP is used instead of a normal delay constant because Desired Requested Work is adjusted almost instantaneously by the workers by adjusting their work rate. Since ideally it should be zero, the desired amount of requested work is just enough to satisfy the demand of the cell.

Effect of Employee Stress on Motivation Table ([(0,1)-(2,2)],(0,1),(0.125, 1.04),(0.25,1.125),(0.375,1.25),(0.5,1.5),(0.625,1.75),(0.75,1.875),(0.875, 1.96),(1,2),(2,2))
Units: Dimensionless
Effect of Employee Stress on Motivation Table represents the relationship between production stress and worker motivation. A

little stress is slightly motivating. Higher levels of stress rapidly hit the maximum motivational effect of stress. The result is an very sharp S-shaped curve.

Effect of Jidoka on Motivation Table ([(0,1)-(2,2)],(0,1),(0.094,1.005),(0.188,1.016),(0.281,1.047),(0.375,1.094),(0.469,1.156),(0.563,1.234),(0.656,1.328),(0.75,1.437),(0.844,1.562),(0.938,1.672),(1.031,1.766),(1.125,1.844),(1.219,1.906),(1.313,1.953),(1.406,1.984),(1.5,2),(2,2))
Units: Dimensionless
Effect of Jidoka on Motivation Table represents the relationship between the rate of implementation of Jidoka and worker motivation. Small success rates generate little enthusiasm. After a certain point higher rate cannot generate much more enthusiasm. The result is an S-shaped curve.

Effect of Motivation On Improvement Process Usage Table ([(0,0)-(1,1)],(0,0),(0.4,0),(0.4625,0.04),(0.525,0.125),(0.5875,0.25),(0.65,0.5),(0.7125,0.75),(0.775,0.875),(0.8375,0.96),(0.95,1),(1,1))
Units: Dimensionless
Effect of Motivation On Improvement Process Usage Table represents the relationship between worker motivation and improvement process usage. Below a certain threshold of motivation the improvement process will not be used. Above the minimum threshold the usage rapidly becomes 100%. The result is an very sharp S-shaped curve.

Effect of Percent Potential Errors on Error Completion Delay Table ([(0,0)-(1,1)],(0,0),(0.125,0.04),(0.25,0.125),(0.375,0.25),(0.5,0.5),(0.625,0.75),(0.75,0.875),(0.875,0.96),(1,1))
Units: Dimensionless
Effect of Percent Potential Errors on Error Completion Delay Table represents the relationship between the percentage of potential errors and the delay they cause. Initial reductions in potential assembly errors have a small effect since their are so many others. Elimination of the last few potential assembly errors also has little effect because they are so rare at that point. The result is an S-shaped curve.

Effect of Percent Requested Work on Cell Production Stress Table ([(0,0)-(1,1)],(0,0),(0.05,0.01),(0.09,0.02),(0.14,0.05),(0.19,0.09),(0.23,0.155),(0.28,0.23),(0.33,0.33),(0.38,0.44),(0.42,0.56),(0.47,0.67),(0.52,0.77),(0.56,0.84),(0.61,0.91),(0.66,0.95),(0.7,0.98),(0.75,1),(1,1))
Units: Dimensionless
Effect of Percent Requested Work on Cell Production Stress Table represents the relationship between production stress and worker motivation. A small backlog produces a small amount of stress. Higher backlogs rapidly produce the maximum level of stress. The result is an very sharp S-shaped curve.

Effect of Process Improvement on Motivation Table ([(-1,0)-(1,2)],(-1,0),(-0.938,0.005),(-0.875,0.016),(-0.813,0.047),(-0.75,0.094),(-0.688,0.156),(-0.625,0.234),(-0.563,0.328),(-0.5,0.437),(-0.438,0.562),(-0.375,0.672),(-0.313

,0.766),(-0.25,0.844),(-0.188,0.906),(-0.125,0.953),(-0.063,0.984),(0,1),(0.063
,1.005),(0.125,1.016),(0.188,1.047),(0.25,1.094),(0.313,1.156),(0.375,1.234
),(0.438,1.328),(0.5,1.437),(0.563,1.562),(0.625,1.672),(0.688,1.766),(0.75
,1.844),(0.813,1.906),(0.875,1.953),(0.938,1.984),(1,2))
Units: Dimensionless
Effect of Process Improvement on Motivation Table represents the
relationship between the rate of implementation of process
improvement and worker motivation. Small success rates generate
little enthusiasm. After a certain point higher rate cannot
generate much more enthusiasm. Loss of improvement is as
demotivating as improvement is motivating. The result is an
S-shaped curve with an inflection point.!

Effectiveness of Improvement Process = Improvement Process Usage * Quality of Improvement Process
Units: Dimensionless
Effectiveness of Improvement Process is the combination of the
usage of the improvement process and its effectiveness.

Employee Stress Motivational Factor = Effect of Employee Stress on Motivation Table
(Production Stress)
Units: Dimensionless
Employee Stress Motivational Factor is the change in the nominal
motivation level due to Production Stress.

Error Elimination Delay = Error Elimination Half Life / 0.7
Units: Week
Error Elimination Delay is the time conversion of the half-life.

Error Elimination Half Life = 12
Units: Week
Error Elimination Half-Life is the time required to reduce the
number of potential errors by 50%.

Error Work Completion Delay = Error Work Completion Delay Factor * Maximum Error Completion Delay
Units: Weeks
Error Work Completion Delay is the portion of the average
assembly delay due to potential assembly errors. (The delay
represents both scrap and rework, small delay --> high
productivity)

Error Work Completion Delay Factor = Effect of Percent Potential Errors on Error Completion Delay Table
(Percent Potential Errors)
Units: Dimensionless
Error Work Completion Delay Factor is the percentage of the
Maximum Error Completion Delay due to the current number of
potential errors.

FINAL TIME = 120
Units: Week
The final time for the simulation.

Improvement Process Usage = Effect of Motivation On Improvement Process Usage Table (Motivation)
Units: Dimensionless
Improvement Process Usage is a measure of the application of the available improvement process. (0% is totally unused, 100% is completely utilized.)

Initial Potential Errors = 50
Units: Errors
Initial Potential Errors is the number of potential assembly errors that can be eliminated using Jidoka.

INITIAL TIME = 0
Units: Week
The initial time for the simulation.

Input = STEP (Step Height , Step Time) + (Pulse Quantity / TIME STEP) * PULSE (Pulse Time , TIME STEP) + RAMP (Ramp Slope , Ramp Start Time , Ramp End Time) + Sine Amplitude * SIN (2 * 3.14159 * Time / Sine Period) + STEP (1, Noise Start Time) * Pink Noise + Table Function (Time / Weeks)
Units: Dimensionless
Input is a dimensionless variable which provides a variety of test input patterns, including a step, pulse, sine wave, and random noise.

Jidoka Implementation Perception Limit = 0.7
Units: Dimensionless
Jidoka Implementation Perception Factor is the percentage of the maximum possible implementation rate beyond which workers cannot perceive incremental change.

Jidoka Implementation Rate = (Potential Errors / Error Elimination Delay) * Effectiveness of Improvement Process
Units: Errors/Week
Jidoka Implementation Rate is the rate at which Systems are put in place to remove potential sources of error. In this model it can only reduce the number of potential errors.

Jidoka Motivational Factor = Effect of Jidoka on Motivation Table (Jidoka Implementation Rate / Maximum Perceivable Jidoka Implementation Rate)
Units: Dimensionless
Jidoka Motivational Factor is the change in the nominal motivation level due to Jidoka improvements.

Maximum Error Completion Delay = 3
Units: Weeks
Maximum Error Completion Delay is the maximum average delay due to the number of potential errors. (The delay represents both scrap and rework, small delay --> high productivity)

Maximum Perceivable Process Improvement Success = ((Nominal Process Work Completion Delay - Minimum Process Work Completion Delay) / Process Improvement Delay) * Process Improvement Perception Limit
Units: Dimensionless
Maximum Perceivable Process Improvement Success is the rate that is perceived as the maximum from a motivational viewpoint. It is the maximum possible improvement rate scaled by the Process Improvement Perception Limit.

Maximum Perceivable Jidoka Implementation Rate = (Initial Potential Errors / Error Elimination Delay) * Jidoka Implementation Perception Limit
Units: Errors/Week
Maximum Perceivable Jidoka Implementation Rate is the rate that is perceived as the maximum from a motivational viewpoint. It is the maximum possible improvement rate scaled by the Jidoka Implementation Perception Limit.

Maximum Work in Progress = 4
Units: Parts
Maximum Work in progress is the upper limit on work that can be in process in the cell at any one time.

Minimum Possible Average Work Completion Delay = Error Work Completion Delay + Process Work Completion Delay
Units: Weeks
Minimum Possible Average Work Completion Delay is the smallest delay possible based on the capabilities of the cell. (The delay represents both scrap and rework, small delay --> high productivity)

Minimum Process Work Completion Delay = 4
Units: Weeks
Minimum Process Work Completion Delay is the minimum possible delay for the particular parts being produced in the cell. Best workers, best cell, etc.) (The delay represents both scrap and rework, small delay --> high productivity)

Motivation = INTEG(Motivational Change , Nominal Motivational Level)
Units: Dimensionless
Work Completion Delay is the average time required to complete one unit of work.

Motivational Change = ((Adjusted Motivational Level - Motivation) / Motivational Change Delay)
Units: Dimensionless/Week
Motivational Change is the rate at which Motivation moves towards the Adjusted Motivational Level..

Motivational Change Delay = Motivational Change Half Life / 0.7
Units: Week
Motivational Change Delay is the time conversion of the half-life.

Motivational Change Half Life = 4
Units: Week
Motivational Entropy Half-Life is the time it takes the
Motivation level to reduce by half.

Noise Correlation Time = 100
Units: Week
The correlation time constant for Pink Noise.

Noise Standard Deviation = 0
Units: Dimensionless
The standard deviation of the pink noise process.

Noise Start Time = 5
Units: Week
Start time for the random input.

Nominal Motivational Level = 0.5
Units: Dimensionless
Nominal Motivational Level is the basic motivation of the
workers to engage in process improvement activities. It is
controlled by many exogenous factors such as layoffs, raises,
praise, etc.

Nominal Process Work Completion Delay = 7
Units: Weeks
Nominal Process Work Completion Delay is the typical delay for
the particular parts being produced in the cell without any
improvement efforts. (Average workers, average cell, etc.) (The
delay represents both scrap and rework, small delay --> high
productivity)

Percent Potential Errors = Potential Errors / Initial Potential Errors
Units: Dimensionless
Percent Potential Errors is the ratio of the current number of
potential assembly errors to the initial number of potential
assembly errors.

Percent Requested Work = Requested Work / (Total Initial Work - Work in Progress)
Units: Dimensionless
Percent Requested Work is the percentage of the total work in
the system that is in the requested work pile.

Pink Noise = INTEG(Change in Pink Noise , 0)
Units: Dimensionless
Pink Noise is first-order autocorrelated noise. Pink noise
provides a realistic noise input to models in which the next
random shock depends in part on the previous shocks. The user
can specify the correlation time. The mean is 0 and the standard
deviation is specified by the user.

Potential Errors = INTEG(- Jidoka Implementation Rate , Initial Potential Errors)
Units: Errors
Potential Errors is a count of the possible mistakes that can be
made in assembly that can be removed by mistake proofing
techniques.

Process Entropy = (Nominal Process Work Completion Delay - Process Work Completion Delay
) / Average Process Entropy Time
Units: Dimensionless
Entropy is the rate at which the Average Work Completion Delay
increases do various factors that can be aggregated into a
generic term called entropy.

Process Entropy Half Life = 10
Units: Weeks
Entropy Half-Life is the time it takes the Process Work Delay
Time to get within 50% of the nominal Process Work Delay Time.
(8 is to low and 12 is to high)

Process Improvement Delay = Process Improvement Half Life / 0.7
Units: Week
Process Improvement Delay is the time conversion of the half-life.

Process Improvement Half Life = 6
Units: Weeks
Process Improvement Half-Life is the time required to reduce the
Process Work Delay Time by 50%.

Process Improvement Motivational Factor = Effect of Process Improvement on Motivation Table
(Process Improvement Success / Maximum Perceivable Process Improvement Success
)
Units: Dimensionless
Process Improvement Motivational Factor is the change in the
nominal motivation level due to process improvements.

Process Improvement Perception Limit = 0.7
Units: Dimensionless
Process Improvement Perception Limitis the percentage of the
maximum possible improvement success beyond which workers cannot
perceive incremental change.

Process Improvement Rate = ((Process Work Completion Delay - Minimum Process Work Completion
Delay
) / Process Improvement Delay) * Effectiveness of Improvement Process
Units: Dimensionless
Process Improvement Rate is the rate at which the Average Work
Completion Delay is reduced due to process improvement
activities.

Process Improvement Success = Process Improvement Rate - Process Entropy
Units: Dimensionless
Process Improvement Success is the ratio of the Process
improvement Rate to the Process Entropy. It is a measure of the
progress felt in a system.

Process Work Completion Delay = INTEG(Process Entropy - Process Improvement Rate
, Nominal Process Work Completion Delay)
Units: Weeks
Work Completion Delay is the average time required to complete
one unit of work. (The delay represents both scrap and rework,
small delay --> high productivity)

Production Stress = Max (Cell Production Stress - Accepted Stress , 0)
Units: Dimensionless
Production Stress is the amount of stress that is currently felt
by the workers for motivational purposes.

Productivity = 1 / Minimum Possible Average Work Completion Delay
Units: Dimensionless/Week

Pulse Quantity = 0
Units: Dimensionless*Week
The quantity to be injected to customer orders, as a fraction of
the base value of Input. For example, to pulse in a quantity
equal to 50% of the current value of input, set to 0.50.

Pulse Time = 5
Units: Week
Time at which the pulse in Input occurs.

Quality of Improvement Process = 0.6
Units: Dimensionless
Quality of Improvement Process is a measure of the methods and
processes used in attempting to make improvements (0% totally
ineffective, 100% is completely effective.)

Ramp End Time = 120
Units: Week
End time for the ramp input.

Ramp Slope = 0
Units: Dimensionless/Week
Slope of the ramp input, as a fraction of the base value (per
week).

Ramp Start Time = 5
Units: Week
Start time for the ramp input.

Requested Work = INTEG(Work Request Rate - Work Start Rate , Work Demand Rate * TIME STEP)
Units: Parts
Requested Work is the amount of work that needs to be done to bring the Completed Work up to the desired level. It is parts as represented by kanban cards that need a part.

Requested Work Adjustment = Max (Requested Work / Desired Requested Work , 1)
Units: Dimensionless
Requested Work Adjustment is used to increase the Desired Average Work Delay to compensate for excessive requested work. It captures the need to work faster to work off the backlog of requested work.

SAVEPER = TIME STEP
Units: Week
The frequency with which output is stored.

Sine Amplitude = 0
Units: Dimensionless
Amplitude of sine wave in customer orders (fraction of mean).

Sine Period = 30
Units: Week
Period of sine wave in customer demand. Set initially to 50 weeks (1 year).

Step Height = 0
Units: Dimensionless
Height of step input to customer orders, as fraction of initial value.

Step Time = 5
Units: Week
Time for the step input.

Stress Acceptance Delay = 12
Units: Week
Stress Acceptance Delay is the average time required for a worker to accept the current production stress level as normal.

Table Function ([(0,0)-(400,2)],(0,1),(3,1),(6,1),(9,1),(12,1),(15,1),(18,1),(21,1),(24,1),(27,1),(30,1),(33,1),(36,1),(39,1),(42,1),(45,1),(48,1),(51,1),(54,1),(57,1),(60,1),(320,1))
Units: Dimensionless

TIME STEP = 0.0625
Units: Week
The time step is for the simulation.

Total Initial Work = 7
Units: Parts
Total Initial Work is the number of kanban cards allotted
to the cell. For initialization the cell is set up to be in
steady state with any excess work allocated to the Completed
Work stock.

Weeks = 1
Units: Weeks

White Noise = Noise Standard Deviation * ((24 * Noise Correlation Time /
TIME STEP) ^ 0.5 * (RANDOM 0 1 ( ) - 0.5))
Units: Dimensionless
White noise input to the pink noise process.

Work Completion Rate = DELAY3i (Work Start Rate , Constrained Desired Average Work Delay
, Work Demand Rate)
Units: Parts/Week
Work Completion Rate is the rate at which parts in progress are
completed.

Work Demand Rate = Work Demand Rate Magnitude * Input
Units: Parts/Week
Work Demand Rate is the product of the Work Demand Rate
Magnitude and the function generator data, input.

Work Demand Rate Magnitude = 0.4
Units: Parts/Week
Work Demand Rate Magnitude is the amplitude of the demand rate
function.

Work in Progress = INTEG(Work Start Rate - Work Completion Rate , Maximum Work in Progress
- (Work Completion Rate * TIME STEP))
Units: Parts
WIP is the number of parts being worked on in the assembly cell.
It consists of partially completed parts and their associated
kanban cards.

Work Removal Rate = MIN (Completed Work / TIME STEP , Work Demand Rate)
Units: Parts/Week
Work Removal Rate is the rate at which completed parts are
removed from the cell.

Work Request Rate = Work Removal Rate
Units: Parts/Week
Work Request Rate is the rate at which new work is requested.
Because of the Kanban system it is equal, by definition, to the
Work Removal Rate.

Work Start Rate = MIN ((Maximum Work in Progress - Work in Progress) , Requested Work) / TIME STEP

Units: Parts/Week

Work Start Rate is the rate at which new work is started. It is equal to the work completion rate. The assumption is made that work can be started immediately. (If WIP is 0, Requested Work is 6, and Maximum WIP is 4, 4 units of work can be moved in one time step so that WIP is 4 and Requested Work is 2.

XII. Appendix B, Project Model

A. Command File

This is the Vensim command file used to generate the data sets used in the analysis of the project model.

```
SPECIAL>LOADMODEL|project.mdl
SIMULATE>READCIN

SIMULATE>SETVAL|Nominal Motivational Level = 0
SIMULATE>RUNNAME|Reference Base
MENU>RUN|o

SIMULATE>RUNNAME|Steady State
MENU>RUN|o

SIMULATE>SETVAL|Nominal Motivational Level = 0.4
SIMULATE>RUNNAME|Low Motivation
MENU>RUN|o

SIMULATE>SETVAL|Nominal Motivational Level = 0.6
SIMULATE>RUNNAME|High Motivation
MENU>RUN|o

SIMULATE>SETVAL|Quality of Improvement Process = 0.4
SIMULATE>RUNNAME|Bad Process
MENU>RUN|o

SIMULATE>SETVAL|Quality of Improvement Process = 0.9
SIMULATE>RUNNAME|Good Process
MENU>RUN|o

SIMULATE>SETVAL|Process Improvement Perception Limit = 0.4
SIMULATE>SETVAL|Jidoka Implementation Perception Limit = 0.4
SIMULATE>RUNNAME|Sensitive Measurement
MENU>RUN|o

SIMULATE>SETVAL|Process Improvement Perception Limit = 1.0
SIMULATE>SETVAL|Jidoka Implementation Perception Limit = 1.0
SIMULATE>RUNNAME|Insensitive Measurement
MENU>RUN|o

SIMULATE>SETVAL|Nominal Motivational Level = 0.4
SIMULATE>SETVAL|Quality of Improvement Process = 0.4
SIMULATE>SETVAL|Process Improvement Perception Limit = 1.0
SIMULATE>SETVAL|Jidoka Implementation Perception Limit = 1.0
SIMULATE>RUNNAME|Failure
MENU>RUN|o

SIMULATE>SETVAL|Nominal Motivational Level = 0.6
```

```
SIMULATE>SETVAL|Quality of Improvement Process = 0.9
SIMULATE>SETVAL|Process Improvement Perception Limit = 0.5
SIMULATE>SETVAL|Jidoka Implementation Perception Limit = 0.5
SIMULATE>RUNNAME|Success
MENU>RUN|o

SIMULATE>SETVAL|Nominal Motivational Level = 1.0
SIMULATE>SETVAL|Quality of Improvement Process = 1.0
SIMULATE>RUNNAME|Perfection
MENU>RUN|o
```

B. Model Documentation

These are the equations used in the project model.

```
Acceptability of Schedule Slip = Acceptability of Schedule Slip Table ( Perceived Percent Complete
        )
        Units: Dimensionless
        Acceptability of Schedule Slip is a measure of the willingness
                        to accept a schedule slip. The normal willingness to slip
                        schedule is assumed to be 1.

Acceptability of Schedule Slip Table ( [(0,0)-(1,1)],(0,0),(0.07,0.03),(0.1
,0.07),(0.161,0.12),(0.2,0.2),(0.25,0.45),(0.3,0.69),(0.335,0.86),(0.36,0.95
),(0.4,1),(0.5,1),(0.75,1),(0.83,1),(0.87,0.92),(0.92,0.63),(0.96,0.31),(0.99
,0),(1,0) )
Units: Dimensionless
Acceptability of Schedule Slip Table represents the willingness
                to slip the completion date on a project as it progresses.

Accepted Completion Time = ( Acceptability of Schedule Slip * Max ( Estimated Completion Time
 - Scheduled Completion Time , 0) ) + Scheduled Completion Time
Units: Month
Accepted Completion Time is the completion time that will be
                accepted after all the dust has cleared.

Actual Percent Complete = Work Done / Initial Planned Work
Units: Dimensionless
Actual Percent Complete is the actual percentage of work that is
                complete.

Actual Productivity = Effect of Experience on Productivity * Effect of Schedule Pressure on Productivity
 * ( Veteran Potential Errors Productivity + Veteran Process Productivity )
Units: Code/(Month*Programmer)
Actual Productivity is the number of Code a single Programmer
                can produce in one Month.

Actual Quality = Effect of Work Quality on Quality * Effect of Schedule Pressure on Quality
 * Effect of Experience on Quality * ( Veteran Potential Errors Quality + Veteran Process Quality)
Units: Dimensionless
Actual Quality is the percentage of work that is performed
                correctly the first time.
```

Adjusted Motivational Level = Employee Stress Motivational Factor * Jidoka Motivational Factor
* Process Quality Improvement Motivational Factor * Process Productivity Improvement Motivational Factor
* Nominal Motivational Level
Units: Dimensionless
Adjusted Motivational Level is the Nominal Motivational Level of
the workers after adjusting for stress and improvement rates.

Average Experience = Total Staff Experience / Staff
Units: Months
Average Experience is the average experience of the staff on the
project.

Average Process Entropy Time = Process Entropy Half Life / 0.7
Units: Month
Average Entropy Time is the time conversion of the half-life.

Average Work Quality = Work Done / Max (0.0001, Undiscovered Rework + Work Done)
Units: Dimensionless
Average Work Quality is the actual average quality of the work
that has been completed.

Best Veteran Process Productivity = 28
Units: Code/(Month*Programmer)
Best Veteran Productivity is the best possible productivity from
a veteran programmer.

Best Veteran Process Quality = 0.7
Units: Dimensionless
Best Veteran Quality is the best possible quality from a veteran
programmer

Completion Percentage = 0.99
Units: Dimensionless
Percentage of Initial Planned Work that must be completed to
consider the project finished.

Cumulative Effort Expended = INTEG(Effort Expended , 0)
Units: Months*Programmers

Cumulative Work Done = INTEG(Rate of Doing Work , 0)
Units: Code
Cumulative Work Done is the total amount of work performed in
order to produce an amount of Work Done equal to Initial Planned
Work.

Decrease in Experience = Out Placement * Average Experience
Units: Months*Programmers/Month
Decrease in Experience is the rate at which total experience is
added to the project.

Desire to Modify Plan = Desire to Modify Plan Table (Percentage of Initial Schedule Used)
Units: Dimensionless
Desire to Modify Schedule is a measure of the desire to adjust
schedule based,

Desire to Modify Plan Table ([(0,0)-(1,1)],(0,0),(0.0513595,0.0394737),(0.0755287,0.0921053),(0.108761,0.263158),(0.135952,0.578947),(0.160121,0.833333),(0.172205,0.912281),(0.199396,0.964912),(0.25,1),(1,1))
Units: Dimensionless
Abandonment of Plan Table captures the relationship between the
amount of initial schedule used and utilization of plan
adjustment equations.

Desired Staffing Level = Project Finished * (Work to Do / (Time Remaining * Perceived Productivity))
Units: Programmers
Number of Programmers required to complete the project in the
scheduled time.

Effect of Experience on Productivity = Effect of Experience on Productivity Table (Experience Ratio)
Units: Dimensionless
Effect of Experience on Productivity reflects the reduced
productivity caused by non veteran Programmers.

Effect of Experience on Productivity Table ([(0,0)-(1,1)],(0,0.2),(0.0392749,0.280702),(0.081571,0.368421),(0.160121,0.482456),(0.253776,0.596491),(0.359517,0.70614),(0.498489,0.802632),(0.613293,0.855263),(0.740181,0.907895),(1,1),(10,1))
Units: Dimensionless
Effect of Experience on Productivity Table reflects the
relationship between the percentage of veterans and
productivity.!

Effect of Experience on Quality = Effect of Experience on Quality Table (Experience Ratio)
Units: Dimensionless
Effect of Experience on Quality reflects the reduced
productivity caused by non veteran Programmers.

Effect of Experience on Quality Table ([(0,0)-(1,1)],(0,0.2),(0.0392749,0.280702),(0.081571,0.368421),(0.160[illegible],0.482456),(0.253776,0.596491),(0.359517,0.70614),(0.498489,0.802632),(0.613[illegible],0.855263),(0.740181,0.907895),(1,1),(10,1))
Units: Dimensionless
Effect of Experience on Quality Table reflects the relationship
between the percentage of veterans and quality.

Effect of Jidoka on Motivation Table ([(0,1)-(2,2)],(0,1),(0.094,1.005),(0.188,1.016),(0.281,1.047),(0.375,1.094),(0.469,1.156),(0.563,1.234),(0.656,1.328),(0.75,1.437),(0.844,1.562),(0.938,1.672),(1.031,1.766),(1.125,1.844),(1.219

,1.906),(1.313,1.953),(1.406,1.984),(1.5,2),(2,2))
Units: Dimensionless
Effect of Jidoka on Motivation Table represents the relationship between the rate of implementation of Jidoka and worker motivation. Small success rates generate little enthusiasm. After a certain point higher rate cannot generate much more enthusiasm. The result is an S-shaped curve.

Effect of Motivation On Improvement Process Usage Table ([(0,0)-(1,1)],(0,0),(0.4,0),(0.4625,0.04),(0.525,0.125),(0.5875,0.25),(0.65,0.5),(0.7125,0.75),(0.775,0.875),(0.8375,0.96),(0.95,1),(1,1))
Units: Dimensionless
Effect of Motivation On Improvement Process Usage Table represents the relationship between worker motivation and improvement process usage. Below a certain threshold of motivation the improvement process will not be used. Above the minimum threshold the usage rapidly becomes 100%. The result is an very sharp S-shaped curve.

Effect of Perceived Schedule Pressure on Motivation Table ([(0,1)-(2,2)],(0,1),(0.125,1.04),(0.25,1.125),(0.375,1.25),(0.5,1.5),(0.625,1.75),(0.75,1.875),(0.875,1.96),(1,2),(2,2))
Units: Dimensionless
Effect of Perceived Schedule Pressure on Motivation Table represents the relationship between schedule pressure and worker motivation. A little stress is slightly motivating. Higher levels of stress rapidly hit the maximum motivational effect of stress. The result is an very sharp S-shaped curve.

Effect of Potential Errors on ProductivityTable ([(0,0.6)-(1,12)],(0,12),(0.125,11.544),(0.25,10.575),(0.375,9.15),(0.5,6.3),(0.625,3.45),(0.75,2.025),(0.875,1.056),(1,0.6))
Units: Code/(Month*Programmer)
Effect of Potential Errors on Productivity Table represents the relationship between the percentage of potential errors and productivity. Initial reductions in potential errors have a small effect since their are so many others. Elimination of the last few potential errors also has little effect because they are so rare at that point. The result is an S-shaped curve.!

Effect of Potential Errors on Quality Table ([(0,0.18)-(1,0.3)],(0,0.3),(0.125,0.2952),(0.25,0.285),(0.375,0.27),(0.5,0.24),(0.625,0.21),(0.75,0.195),(0.875,0.1848),(1,0.18))
Units: Dimensionless
Effect of Potential Errors on Quality Table represents the relationship between the percentage of potential errors and quality. Initial reductions in potential errors have a small effect since their are so many others. Elimination of the last few potential errors also has little effect because they are so rare at that point. The result is an S-shaped curve.!

Effect of Process Improvement on Motivation Table ([(-1,0)-(1,2)],(-1,0),(
-0.938,0.005),(-0.875,0.016),(-0.813,0.047),(-0.75,0.094),(-0.688,0.156),(-
0.625,0.234),(-0.563,0.328),(-0.5,0.437),(-0.438,0.562),(-0.375,0.672),(-0.313
,0.766),(-0.25,0.844),(-0.188,0.906),(-0.125,0.953),(-0.063,0.984),(0,1),(0.063
,1.005),(0.125,1.016),(0.188,1.047),(0.25,1.094),(0.313,1.156),(0.375,1.234
),(0.438,1.328),(0.5,1.437),(0.563,1.562),(0.625,1.672),(0.688,1.766),(0.75
,1.844),(0.813,1.906),(0.875,1.953),(0.938,1.984),(1,2))
Units: Dimensionless
Effect of Process Improvement on Motivation Table represents the
relationship between the rate of implementation of process improvement and worker motivation. Small success rates generate little enthusiasm. After a certain point higher rate cannot generate much more enthusiasm. Loss of improvement is as demotivating as improvement is motivating. The result is an S-shaped curve with an inflection point.!

Effect of Schedule Pressure on Productivity = Effect of Schedule Pressure on Productivity Table
(Perceived Schedule Pressure)
Units: Dimensionless
Effect of Schedule Pressure on Productivity reflects the reduced
productivity caused by non veteran Programmers.

Effect of Schedule Pressure on Productivity Table ([(0,0)-(2,2)],(0,0.5),(
0.163142,0.517544),(0.344411,0.552632),(0.537764,0.622807),(0.785498,0.789474
),(1,1),(1.19637,1.26316),(1.38369,1.62281),(1.52266,1.84211),(1.6435,1.92982
),(1.8006,1.97368),(2,2),(10,2))
Units: Dimensionless
Effect of Schedule Pressure on Productivity Table reflects the
relationship between the percentage of veterans and productivity.

Effect of Schedule Pressure on Quality = Effect of Schedule Pressure on Quality Table
(Perceived Schedule Pressure)
Units: Dimensionless
Effect of Schedule Pressure on Quality reflects the reduced
quality speeding up the work.

Effect of Schedule Pressure on Quality Table ([(0,0)-(10,1)],(0,1),(1,1),(
1.12085,0.982456),(1.22356,0.949561),(1.31722,0.899123),(1.5,0.75),(1.69789
,0.585526),(1.79758,0.535088),(1.89728,0.508772),(2,0.5),(10,0.5))
Units: Dimensionless
Effect of Schedule Pressure on Quality Table reflects the
relationship between schedule pressure and quality

Effect of Work Quality on Quality = Effect of Work Quality on Quality Table
(Average Work Quality)
Units: Dimensionless
Effect of Work Quality on Quality is the percentage of current
work done correctly that is not based on previously done work that is actually undiscovered rework.

Effect of Work Quality on Quality Table ([(0,0)-(1,1)],(0,0.1),(0.1,0.25),
(0.2,0.35),(0.3,0.45),(0.4,0.55),(0.5,0.65),(0.6,0.74),(0.7,0.83),(0.3,0.9)
,(0.9,0.95),(1,1))
Units: Dimensionless
Effect of Work Quality on Quality Table represents the effect of
undiscovered rework on the quality of current work.

Effectiveness of Improvement Process = Improvement Process Usage * Quality of Improvement Process
Units: Dimensionless
Effectiveness of Improvement Process is the combination of the
usage of the improvement process and its effectiveness.

Effort Expended = Staff * Project Finished
Units: Programmers

Employee Stress Motivational Factor = Effect of Perceived Schedule Pressure on Motivation Table
(Perceived Schedule Pressure)
Units: Dimensionless
Employee Stress Motivational Factor is the change in the nominal
motivation level due to Production Stress.

Error Elimination Delay = Error Elimination Half Life / 0.7
Units: Month
Error Elimination Delay is the time conversion of the half-life.

Error Elimination Half Life = 12
Units: Month
Error Elimination Half-Life is the time required to reduce the
number of potential errors by 50%.

Estimated Completion Time = IF THEN ELSE (Project Finished = 0, Scheduled Completion Time
, Required Time to Finish + Time)
Units: Month
Estimated Completion Time is the time the project should finish
based on the current state of the project.

Expected Work Accomplishment = Max (Perceived Productivity * Planning Staff
, Minimum Expected Work Accomplishment)
Units: Code/Month
Expected Work Accomplishment is the work rate based on current
staff and productivity.

Experience Ratio = Min (1, Average Experience / Months per Veteran)
Units: Dimensionless
Ratio of veteran to non veteran Programmers.

FINAL TIME = 100
Units: Month
The final time for the simulation.

Hiring = Max ((Desired Staffing Level - Staff) / Hiring Delay , 0)
Units: Programmers/Month
Hiring is the rate at which Programmers are added to the staff.

Hiring Delay = 2
Units: Month
Hiring Delay is the average time required to hire new staff.

Improvement Process Usage = Effect of Motivation On Improvement Process Usage Table (Motivation)
Units: Dimensionless
Improvement Process Usage is a measure of the application of the available improvement process. (0% is totally unused, 100% is completely utilized.)

Increase in Experience = (Hiring * New Hire Experience) + On the Job Gains
Units: Months*Programmers/Month
Increase in Experience is the rate at which total experience is added to the project.

Initial Average Experience = 1
Units: Month
Initial Average Experience is the experience per Programmer on the project at the beginning. (It might be non-zero if the project is a repeat or similar to other projects.)

Initial Perceived Schedule Pressure = 1
Units: Dimensionless
Initial Perceived Schedule Pressure is the schedule pressure felt at the start of the project.

Initial Planned Work = 2000
Units: Code
Initial Planned Work is the total amount of work initially planned for the project.

Initial Potential Errors = 50
Units: Errors
Initial Potential Errors is the number of potential assembly errors that can be eliminated using Jidoka.

Initial Schedule Completion Time = 50
Units: Month
Initial Schedule Completion Time is the originally planned project completion date.

Initial Staff = 10
Units: Programmers
Initial number of Programmers on project.

INITIAL TIME = 0
Units: Month
The initial time for the simulation.

Initially Planned Staff = 10
Units: Programmers
Initially Planned Staff is the number of Martians initially
thought to be required for the project.

Jidoka Implementation Perception Limit = 0.7
Units: Dimensionless
Jidoka Implementation Perception Factor is the percentage of the
maximum possible implementation rate beyond which workers cannot
perceive incremental change.

Jidoka Implementation Rate = ((Potential Errors / Error Elimination Delay
) * Effectiveness of Improvement Process) * Project Finished
Units: Errors/Month
Jidoka Implementation Rate is the rate at which Systems are put in
place to remove potential sources of error. In this model it can
only reduce the number of potential errors.

Jidoka Motivational Factor = Effect of Jidoka on Motivation Table (Jidoka Implementation Rate
/ Maximum Perceivable Jidoka Implementation Rate)
Units: Dimensionless
Jidoka Motivational Factor is the change in the nominal
motivation level due to Jidoka improvements.

Maximum Perceivable Process Productivity Improvement Success = ((Best Veteran Process Productivity
- Nominal Veteran Process Productivity) / Process Improvement Delay) * Process Productivity
Improvement Perception Limit
Units: Code/(Month*Month*Programmer)
Maximum Perceivable Process Improvement Success is the rate that
is perceived as the maximum from a motivational viewpoint. It is
the maximum possible improvement rate scaled by the Process
Improvement Perception Limit.

Maximum Perceivable Process Quality Improvement Success = ((Best Veteran Process Quality
- Nominal Veteran Process Quality) / Process Improvement Delay) * Process Quality Improvement
Perception Limit
Units: Dimensionless/Month
Maximum Perceivable Process Improvement Success is the rate that
is perceived as the maximum from a motivational viewpoint. It is
the maximum possible improvement rate scaled by the Process
Improvement Perception Limit.

Maximum Perceivable Jidoka Implementation Rate = (Initial Potential Errors
/ Error Elimination Delay) * Jidoka Implementation Perception Limit
Units: Errors/Month
Maximum Perceivable Jidoka Implementation Rate is the rate that is perceived as the maximum from a
motivational viewpoint. It is the maximum possible improvement rate scaled by the
Jidoka Implementation Perception Limit.

Minimum Completion Time = 2
Units: Month
Planned Completion Time is the originally planned project
completion time.

Minimum Expected Work Accomplishment = 0.001
Units: Code/Month
Minimum Expected Work Accomplishment is used to keep Expected
Work Accomplishment Rate from umderflowing.

Months per Month = 1
Units: Months/Month
Months per Month is the number of months in a month. It is
required to make sure the units are correct.

Months per Veteran = 24
Units: Months
Months per Veteran is the number of months required to turn a
rookie into a veteran.

Motivation = INTEG(Motivational Change , Nominal Motivational Level)
Units: Dimensionless
Work Completion Delay is the average time required to complete
one unit of work.

Motivational Change = ((Adjusted Motivational Level - Motivation) / Motivational Change Delay
)
Units: Dimensionless/Month
Motivational Change is the rate at which Motivation moves
towards the Adjusted Motivational Level..

Motivational Change Delay = Motivational Change Half Life / 0.7
Units: Month
Motivational Change Delay is the time conversion of the half-life.

Motivational Change Half Life = 4
Units: Month
Motivational Entropy Half-Life is the time it takes the
Motivation level to reduce by half.

New Hire Experience = 1
Units: Month
New Hire Experience is the amount of experience each new hire
brings to the project. (It might be non-zero if the project is a
repeat or similar to other projects.)

Nominal Motivational Level = 0.5
Units: Dimensionless
Nominal Motivational Level is the basic motivation of the
workers to engage in process improvement activities. It is
controlled by many exogenous factors such as layoffs, raises,
praise, etc.

Nominal Veteran Process Productivity = 1.4
Units: Code/(Month*Programmer)
Nominal Veteran Productivity is the nominal productivity from a
veteran programmer.

Nominal Veteran Process Quality = 0.42
Units: Dimensionless
Best Veteran Quality is the nominal quality from a veteran
programmer.

On the Job Gains = Staff * Months per Month
Units: Months*Programmers/Month
On the Job Gains is the experience gained by being on the
project.

Out Placement = Max ((Staff - Desired Staffing Level) / Placement Delay
, 0)
Units: Programmers/Month
Out Placement is the rate at which Programmers are added to the
project.

Perceived Percent Complete = (1 - Work to Do / Initial Planned Work)
Units: Dimensionless
Perceived Percent Complete is the percent of work that is
thought to be complete.

Perceived Productivity = IF THEN ELSE (Cumulative Effort Expended > 0, Work Perceived Complete
/ Cumulative Effort Expended , Actual Productivity)
Units: Code/(Month*Programmer)
Perceived Productivity is the perceived level of productivity
for the project.

Perceived Schedule Pressure = smooth3i (Schedule Pressure , Time to Feel Pressure
, Initial Perceived Schedule Pressure)
Units: Dimensionless
Schedule Pressure is the ratio of the allowed to finish the
project and the time required to finish the project.

Percent Potential Errors = Potential Errors / Initial Potential Errors
Units: Dimensionless
Percent Potential Errors is the ratio of the current number of
potential assembly errors to the initial number of potential
assembly errors.

Percentage of Initial Schedule Used = Min (1, Time / Initial Schedule Completion Time)
Units: Dimensionless
Percentage of Initial Schedule Used is a measure of the time
spent on the project vs. the originally scheduled time.

Placement Delay = 1
Units: Month
Placement Delay is the average time required to find new work for staff.

Planning Staff = (Desire to Modify Plan * Staff) + ((1 - Desire to Modify Plan) * Initially Planned Staff)
Units: Programmers
Planning Staff is the staff level used for planning purposes.

Potential Errors = INTEG(- Jidoka Implementation Rate , Initial Potential Errors)
Units: Errors
Potential Errors is a count of the possible mistakes that can be made in assembly that can be removed by mistake proofing techniques.

Potential Work Rate = Actual Productivity * Staff
Units: Code/Month
Potential Work Rate is the highest possible rate of work as determined by the number of Programmers and the average productivity per Programmer.

Process Entropy Half Life = 10
Units: Months
Entropy Half-Life is the time it takes the Process Work Delay Time to get within 50% of the nominal Process Work Delay Time. (8 is to low and 12 is to high)

Process Improvement Delay = Process Improvement Half Life / 0.7
Units: Month
Process Improvement Delay is the time conversion of the half-life.

Process Improvement Half Life = 6
Units: Months
Process Improvement Half-Life is the time required to reduce the Process Work Delay Time by 50%.

Process Productivity Entropy = ((Veteran Process Productivity - Nominal Veteran Process Productivity) / Average Process Entropy Time) * Project Finished
Units: Code/(Month*Month*Programmer)
Entropy is the rate at which the Average Work Completion Delay increases do various factors that can be aggregated into a generic term called entropy.

Process Productivity Improvement Motivational Factor = Effect of Process Improvement on Motivation Table
(Process Productivity Improvement Success / Maximum Perceivable Process Productivity Improvement Success)
Units: Dimensionless
Process Productivity Improvement Motivational Factor is the change in the nominal motivation level due to process productivity improvements.

Process Productivity Improvement Perception Limit = 0.7
Units: Dimensionless
Process Improvement Perception Limits the percentage of the
maximum possible improvement success beyond which workers cannot
perceive incremental change.

Process Productivity Improvement Rate = (((Best Veteran Process Productivity
- Veteran Process Productivity) / Process Improvement Delay) * Effectiveness of Improvement Process
) * Project Finished
Units: Code/(Month*Month*Programmer)
Process Improvement Rate is the rate at which the Average Work
Completion Delay is reduced due to process improvement
activities.

Process Productivity Improvement Success = Process Productivity Improvement Rate
- Process Productivity Entropy
Units: Code/(Month*Month*Programmer)
Process Improvement Success is the ratio of the Process
improvement Rate to the Process Entropy. It is a measure of the
progress felt in a system.

Process Quality Entropy = ((Veteran Process Quality - Nominal Veteran Process Quality
) / Average Process Entropy Time) * Project Finished
Units: Dimensionless/Month
Entropy is the rate at which the Average Work Completion Delay
increases do various factors that can be aggregated into a
generic term called entropy.

Process Quality Improvement Motivational Factor = Effect of Process Improvement on Motivation Table
(Process Quality Improvement Success / Maximum Perceivable Process Quality Improvement Success
)
Units: Dimensionless
Process Quality Improvement Motivational Factor is the change in
the nominal motivation level due to process quality
improvements.

Process Quality Improvement Perception Limit = 0.7
Units: Dimensionless
Process Improvement Perception Limits the percentage of the
maximum possible improvement success beyond which workers cannot
perceive incremental change.

Process Quality Improvement Rate = (((Best Veteran Process Quality - Veteran Process Quality
) / Process Improvement Delay) * Effectiveness of Improvement Process) *
Project Finished
Units: Dimensionless/Month
Process Improvement Rate is the rate at which the Average Work
Completion Delay is reduced due to process improvement
activities.

Process Quality Improvement Success = Process Quality Improvement Rate - Process Quality Entropy
Units: Dimensionless/Month
Process Improvement Success is the ratio of the Process
improvement Rate to the Process Entropy. It is a measure of the
progress felt in a system.

Project Finished = IF THEN ELSE ((Initial Planned Work * Completion Percentage
) >= Work Done , 1, 0)
Units: Dimensionless

Quality of Improvement Process = 0.6
Units: Dimensionless
Quality of Improvement Process is a measure of the methods and
processes used in attempting to make improvements (0% totally
ineffective, 100% is completely effective.)

Rate of Doing Work = Rework Generation + Work Accomplishment
Units: Code/Month
Rating of Doing Work is the rate at which Work to Do is turned
into something else.

Requested Work = 1
Units: Parts
Requested Work is the amount of work that needs to be done to
bring the Completed Work up to the desired level. It is parts as
represented by kanban cards that need a part.

Required Time to Finish = Project Finished * (Work to Do / Expected Work Accomplishment
)
Units: Month
Required Time to Finish is how much longer the project should
run based on current work, staffing and productivity.

Rework Discovery = Undiscovered Rework / Time to Discover Rework
Units: Code/Month
Rework Discovery is the rate at which Undiscovered Rework is
turned into known Work to Do. In other words the rate at which
rework is discovered.

Rework Generation = Project Finished * (1 - Actual Quality) * Min (Potential Work Rate
, (Work to Do / TIME STEP))
Units: Code/Month
Rework Generation is the rate at which Work to Do is turned into
Undiscovered Rework.

SAVEPER = TIME STEP
Units: Month
The frequency with which output is stored.

Schedule Change Delay = 1
Units: Month
Schedule Change Delay is the average time between schedule
revisions.

Schedule Modification = (Accepted Completion Time - Scheduled Completion Time
) / Schedule Change Delay
Units: Dimensionless
Schedule Modification is the change in schedule due changes in
the Estimated Completion Time.

Schedule Pressure = Required Time to Finish / Time Remaining
Units: Dimensionless
Schedule Pressure is the ratio of the allowed to finish the
project and the time required to finish the project.

Scheduled Completion Time = INTEG(Schedule Modification , Initial Schedule Completion Time
)
Units: Month
Scheduled Completion Time is the time at which the project is
planned to end.

Staff = INTEG(Hiring - Out Placement , Initial Staff)
Units: Programmers
Staff is the total number of Programmers on the project.

Time Remaining = Max (Scheduled Completion Time - Time , Minimum Completion Time
)
Units: Month
Time Remaining is the time left in the project based on the
original plan.

TIME STEP = 0.0625
Units: Month
The time step for the simulation.

Time to Discover Rework = 6
Units: Month
Time to Discover Rework is the average time required to discover
rework.

Time to Feel Pressure = 1
Units: Months
Time to Feel Pressure is the delay between schedule pressure
being applied and being felt.

Total Initial Work = 7
Units: Parts
Total Initial Work is the number of kanban cards allotted
to the cell. For initialization the cell is set up to be in
steady state with any excess work allocated to the Completed
Work stock.

Total Staff Experience = INTEG(Increase in Experience - Decrease in Experience
, Staff * Initial Average Experience)
Units: Months*Programmers
Total Staff Experience is the cumulative experience of the staff
on the project.

Undiscovered Rework = INTEG(Rework Generation - Rework Discovery , 0)
Units: Code
Undiscovered Rework is the amount of work that has been done
that must be redone.

Veteran Potential Errors Productivity = Effect of Potential Errors on ProductivityTable
(Percent Potential Errors)
Units: Code/(Month*Programmer)
Veteran Potential Errors Productivity is the reduction in
quality associated with the percentage of potential errors.

Veteran Potential Errors Quality = Effect of Potential Errors on Quality Table
(Percent Potential Errors)
Units: Dimensionless
Effect of Potential Errors on Quality is the reduction in
quality associated with the percentage of potential errors.

Veteran Process Productivity = INTEG(Process Productivity Improvement Rate
- Process Productivity Entropy , Nominal Veteran Process Productivity)
Units: Code/(Month*Programmer)
Veteran Productivity is the average productivity level for a
veteran worker.

Veteran Process Quality = INTEG(Process Quality Improvement Rate - Process Quality Entropy
, Nominal Veteran Process Quality)
Units: Dimensionless
Veteran Quality is the average quality level for a veteran
worker.

Work Accomplishment = Project Finished * Actual Quality * Min (Potential Work Rate
, (Work to Do / TIME STEP))
Units: Code/Month
Work Accomplishment is the rate at which Work to Do is converted
into Work Done.

Work Done = INTEG(Work Accomplishment , 0)
Units: Code
Work Done is the total amount of work completed for the project.

Work in Progress = 1
Units: Parts
WIP is the number of parts being worked on in the assembly cell.
It consists of partially completed parts and their associated
kanban cards.

Work Perceived Complete = Initial Planned Work - Work to Do
Units: Code
Work Perceived Complete is the amount of work thought to be
complete.

Work to Do = INTEG(Rework Discovery - Rework Generation - Work Accomplishment
, Initial Planned Work)
Units: Code
Work to do is the total amount of work that has been identified
but not yet accomplished.