Using System Dynamics in Business Simulation Training Games

by

Jennifer Ching-Wen Han

Submitted to the Department of Electrical Engineering and Computer Science in Partial Fulfillment of the Requirements for the Degree of

Masters of Engineering in Electrical Engineering and Computer Science

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

May 9th, 1997

Author .....
Department of Electrical Engineering and Computer Science
May 9th, 1997

Certified by .....
Richard C. Larson
Thesis Supervisor

Accepted by .....
Arthur C. Smith
Chairman, Department Committee on Graduate Theses

Using System Dynamics in Business Simulation Training Games

by

Jennifer Ching-Wen Han

Submitted to the Department of Electrical Engineering and Computer Science

May 9th, 1997

In Partial Fulfillment of the Requirements for the Degree of Masters of Engineering in

Electrical Engineering and Computer Science

ABSTRACT

This thesis project includes researching and designing a system dynamics model for use in an existing Andersen Consulting Education (ACE) project management training game. This serves two major purposes. The first is to show how system dynamics can be used as a realistic and potentially superior method of business modeling. The second is to actually improve the existing game. The current training game, Project Management Fundamentals School, leads trainees through managing a project via a series of case studies. The improved game will include a computerized system dynamics model so that trainee inputs can effect the outcome of the game. This change will allow players to test their suggestions, making the game more interactive and interesting. The base system dynamics model is from Tarek Abdel-Hamid's 1984 Ph.D. thesis "The Dynamics of Software Development Project Management: An Integrative System Dynamics Approach." This model was ported from DYNAMO, an outdated system dynamics modeling software package, to Vensim, a newer modeling software package which is compatible with many others on the current market. The Abdel-Hamid model is well tested and accepted. However it does not meet all of ACE's requirements for their game engine, PERIOD1. ACE sent a list of initial design suggestions for this engine. These potential variable and structural changes to the base model will be discussed in this thesis.

Thesis Supervisor: Richard C. Larson
Title: Professor of Electrical Engineering

Acknowledgments

This thesis is dedicated to my parents Shih-Ping and Winnie Han. I would not have been able to make it through these five years without their love, encouragement, and confidence in me.

I would like to thank my thesis advisor Professor Richard Larson for his support on my thesis project. He presented me with many opportunities to learn new skills throughout this project - from supervising undergraduate students to giving presentations to managing my time more effectively. I sincerely appreciate all of his feedback regarding my thesis and my ACE and MasterWorks presentations.

I would also like to thank my advisors Professor Steven Lerman and Professor John Sterman, who have both helped me considerably on my thesis.

I would like to thank Tom Long of Andersen Consulting Education for his continued interest and funding of this project. I especially appreciate his invitations to St. Charles to meet and present to ACE professionals. These opportunities have improved my communication skills and increased my confidence in giving professional presentations.

I would also like to thank my UROP students, Autumn Steuckrath and Eugene Fung. They have both contributed significantly toward this project. I would especially like to thank Eugene for creating the figures for my thesis on such short notice.

I would like to thank Scott Rockart for his help regarding Vensim and DYNAMO.

Finally, I would like to thank the following people, who have made my five years at MIT a much more enjoyable and interesting experience:

Alice and Sophia Han - For being the best sisters ever and showing me that family is forever. Jennifer Liu - For being my best friend since kindergarten. Scott Van Woudenberg - For having so much confidence in me and being someone I can always turn to. Julie Kissel, Fuyiu Yip - For being my friends since high school! Natalie Burger - For being my roommate and co-social chair. For sharing so much of MIT with me. Paolo Narvaez, Craig Barrack, Carol Lee - For making my years at New House so much more fun. Lisa Hsieh - For following me everywhere...and giving me pressure to set a good example!!! Thomas Burbine - For proofreading my thesis twice. For a really good time in Ashdown. Benjie Sun - For stressing about our thesis together. And teaching me what could happen if I procrastinated.

Table of Contents

List of Figures

1. Introduction

1.1 General Background

This Masters of Engineering thesis project involves the work of MIT professors, MIT students, and Andersen Consulting Education (ACE). Andersen Consulting Education is currently interested in improving its training process. They are looking to decrease costs, increase the number of employees trained, and improve current modeling techniques. MIT and ACE have jointly agreed to research the possibility of a distributed, multi-user business simulation game using system dynamics modeling techniques.

There are three professors from MIT working on this project with ACE, each serving as an expert in his specific field. Professor Richard C. Larson is the Director of the Center for Advanced Educational Services (CAES). Professor Steven R. Lerman is the Director of Center for Educational Computing Initiatives (CECI). Professor John D. Sterman is the Director of the System Dynamics Group at the Sloan School of Management. There are also two undergraduate students and one graduate student from MIT involved in this project.

1.2 Overview of Project

This project focuses on ACE's goal of improving current modeling techniques. In preparation, an intensive study on existing business simulation games was done. Computerized business simulation games from the 1960s through the1990s were found using resources such as the World Wide Web, books, articles, and game manuals. An annotated bibliography was created using informa-

tion collected, such as the goals of the game, the time it takes to play the game, and the number of players needed. This bibliography will allow Andersen employees to look for existing business simulations and to obtain a quick summary of the game. The bibliography is currently in HTML format and on a MIT based web page (http://www-caes.mit.edu/.) The web page allows Andersen employees to run searches on games based on type (general, functional, industry specific), year, title, and authors. Furthermore, the web page will allow the public to e-mail CAES if they know of any games not included in the current bibliography. Members of CAES (either a graduate research assistant or undergraduate student) will review the new game and insert the game, if accepted, into the annotated bibliography. This is a relatively easy way to maintain the web page and keep the bibliography updated. Since Andersen will most likely be interested in the latest and greatest games, the ability to dynamically add new games is very valuable.

Nine of the games in the bibliography were chosen to be researched in depth. The games which were chosen are all created in the 1990s and contain fairly sophisticated engines. A summary report of these games was compiled, containing information such as the company who published the game and the retail cost of the game. ACE would then be able to contact these companies if there is an interest in the game engine.

This thesis explores the possibility of utilizing system dynamics to model major components of project management in an existing ACE training game. This serves two main purposes. The first is to show how system dynamics can be used as a realistic and potentially superior method of business modeling. The second is to actually improve the existing game. The current training game, Project Management Fundamentals School, leads trainees through managing a project via a series

of case studies. Each of the first three cases represents different decisions made by project managers such as headcount, budget, quality level, and schedule. In these three case studies, trainees analyze and discuss why certain parameters are unrealistic and how they should be changed. The trainees, however, will not be able to test their suggested changes. The improved game will include a computerized system dynamics model so that trainees will be able to input their own parameters. This change will allow players to test their suggestions, making the game more interactive and interesting.

It must be noted that the first draft of the system dynamics model of project management is pedagogical. Real data often takes years to obtain. Also, the game should not be used for actual training purposes until it has gone through extensive testing for accuracy. Training students on a faulty model can be extremely harmful and dangerous.

1.3 Outline of Thesis

This thesis will begin with background information on business simulation games and system dynamics modeling. Next, it will describe how system dynamics can be successfully used to model project management. The main part of this thesis will focus on describing the existing Project Management Fundamentals School training process, and the initial design of the new computerized system dynamics model. The final section gives suggestions regarding the future steps and goals of this project.

2. Business Simulation Games

2.1 Brief History of Business Simulation Games

The first business simulation games were introduced in the late 1950s, contributed by developments in war games, operational research, computer technology, and education theory [1]. In the early 1960s, some business games such as the Carnegie Tech Management Game and INTOP were widely available. By 1968, virtually all business schools were using at least some form of gaming in their teaching programs [2]. The purpose of these educational simulation games was to help students learn to apply the knowledge they had gained in the classroom to actual business problems. As soon as the games became available, companies realized their potential in training employees. By 1970, it was estimated that over 200 games were in existence and over 100,000 executives had been exposed to them [2].

There are two main purposes for business simulation games - education and research [2]. Business simulations have been used to help students apply their conceptual knowledge to business problems. Business simulation games have also been used in research to analyze decision-making processes. Since this project uses simulations for training purposes, the rest of this paper will concentrate on using business simulation games for education.

2.2 Benefits of Business Simulation Games

There are many benefits to training employees and students using business simulation games. Business simulation games allow players to make decisions without risks to any companies, per-

sons, or environments. This gives the player a chance to gain experience and learn about the possible outcomes prior to actually implementing a potentially bad idea [3]. Furthermore, business simulation games provide rapid feedback, allowing the player to see the effect of his decisions more clearly [4]. In the real world, delays make it difficult for people to link causes and effects. Another advantage of business simulation games is being able to play multiple times. Often times in industry, an employee has only one chance to make a decision. Trying different paths will help a trainee test his mental model of the business world.

Many advocates of business simulation games state that the greatest asset of simulations is the power of immediate feedback. The time it takes from the day a business decision is made to the day feedback is received can easily be over a year. This delay makes it difficult for people to link the cause of an effect to their decisions. One marketing executive cites that his project managers routinely "assuage short-term pain" by cutting prices to make sales, only to see retail inventories climb and then drop steeply a few months later [5]. Simulation games can help these managers see that their short term gains are causing the long term disasters.

2.3 Typical Games and Examples

In a typical business simulation game, players have to analyze situations, identify problems and opportunities, and make decisions over a number of iterations composed of simulated months, quarters, and/or years.

Beyond trying to reproduce the numbers in business (profits, revenues, losses), many successful software simulations try to reproduce human frailties and unexpected events that complicate working life [5]. Examples include losing work days due to sickness and sudden increases in vendor prices. The most realistic simulations also include workplace personalities which range from a workaholic who feels she doesn't get enough credit to an ornery veteran who is no longer motivated to work hard.

Simulations have been used in many situations to correct an employee or student's faulty mental model of business dynamics. The Planning and Control Inc. (PCI) is a New York based training firm which has been commended by many of its clients. A process control engineer at GE was involved in one of PCI's leadership training programs. The PCI software challenged his team of GE employees to finish a project within a certain budget and deadline. His team's inclination was to add more people into the project, assuming that more people would finish the project faster. The simulation showed, however, that the extra workers took longer to learn the job and caused more communication problems. A GE project manager commented that when this happens in a real plant or office, managers rationalize it by thinking they added the wrong people [5]. As another example, a properties claim manager noticed that managers who "skimp on the claim process" and get through paperwork quickly with low headcount were rewarded with promotions. A simulation he used showed that as a result, months later, his company was making higher payments months on claims which were not adequately investigated [5].

A common phenomenon was mentioned by John Sterman, the Director of the System Dynamics Group at the MIT Sloan School of Management. A manager may make a decision which leads to

small short term gains, but huge losses after a delay of a few quarters. The manager makes the decision, and before the delay is over, he is promoted to a new position within the company. The employee who takes his previous position will then be blamed for the huge losses caused by the original decision. If no one notices this effect, the bad decision maker will be continuously promoted, leaving a trail of disastrous events behind him and potentially leading to the downfall of the entire company. Business simulation games can compress space and time in a realistic way so that people can see the ramifications of their decisions.

The People Express simulation, created by John Sterman, is another business game which has been commended by multiple sources [6, 7]. The game allows students to take over an airline company and make decisions such as hiring, buying more airplanes, and changing the fares. The original company strategy included intensive training and rapid growth. This led the company to bankruptcy because a high percentage of the employees were inexperienced and too much time was spent on training [8]. Business games such as this one gives students a chance to explore the consequences of various strategies and gain experience without risking a real company.

There are many effective business simulation games, such as the ones mentioned in this section of the paper, used around the world. Some of these games are included in the annotated bibliography web page for ACE. The bibliography will be updated as new games are found, hence maintaining its value to ACE.

2.4 Effectiveness of Business Simulation Games

A number of studies have been aimed at measuring the effectiveness of simulation games. A few internal validity studies have been conducted, comparing learning from business games to learning from case studies. Superior results were found for students playing the games in terms of course grades, performance on concepts examinations, goal-setting exercises, basic management skills, and management ethics [1]. The external validity of business games have shown a strong correlation between successful executive game play and career success [1].

Ultimately, however, the effectiveness of the game depends on the quality of the simulation in representing the behavior of the real world [1]. By the very definition, a model eliminates some details of the actual business environment. A good model will contain all the main components of the business sector simulated, while eliminating extraneous details [1]. A bad model is very dangerous since it will teach and reinforce unrealistic business models. Therein lies a major difficulty in creating a good business simulation game. There are many different ways to model a business environment. In the next section, modeling and system dynamics, a relatively new way to model, will be discussed and assessed.

3. Modeling and System Dynamics

3.1 Modeling

Before delving into a description of system dynamics modeling, it is important to understand why computerized modeling is used.

Computerized models are important because there are limitations to mental models. People use mental models everyday to make decisions. Decisions and actions are based not on the actual state of the environment, but on mental images of the state of the world and how the various parts of the system are related [9]. Humans consistently make suboptimal decisions when faced with complex systems [4] because they are not adept at interpreting the assumptions of their own mental models and are unable to correctly infer any but the most simplistic causal maps. This is due to "bounded rationality" - the human limitations of attention, memory, recall, and information processing given the time constraints imposed [4]. Furthermore, people are effected by organizational context, authority relations, peer pressure, cultural perspective, and selfish motives [9]. Computerized models, on the other hand, are not biased and can infallibly compute the logical sequences of the system, given the modeler's assumptions. Another advantage to using computerized models is that they are explicit and their assumptions can be reviewed by multiple people [9]. This increases the probability that the base assumptions are correct. System dynamics modeling encompasses all these advantages.

3.2 Founder of System Dynamics

Jay Forrester is credited as the founder of system dynamics. Forrester went to the Massachusetts Institute of Technology (MIT) for graduate school, where he majored in electrical engineering. While at MIT, he invented magnetic core memory, which became the dominant way to store data in a computer for about twenty years. In the late fifties, he decided to learn more about management and enrolled in MIT's Sloan School of Management. During this time, he talked to many managers from major corporations who were puzzled about the large fluctuations in variables such as inventory and work force level needed within their organization. For example, General Electric did not understand why their household appliance plants in Kentucky sometimes worked at full capacity with overtime and then two or three years later, needed to lay off half their employees because there was not enough work for everyone. Forrester set out to simulate the situation using variables such as inventory, number of employees, and production rate. This first inventory control system was the beginning of system dynamics [10].

3.3 Main Components of System Dynamics

A dynamic system is a system which changes with the progress of time. The components in a dynamics system interact to create a progression of system conditions. The main features of a system dynamics model are stocks, flows, and feedback loops. The following paragraphs will explain each of these components in more detail. It is important to note that the concepts of system dynamics stem from control theory. Stocks, for example, are analogous to system state variables in engineering.

3.3.1 Stocks and Flows

Stocks, also referred to as 'levels,' are the accumulations within the system. Flows, also referred to as 'rates,' are simply the rate at which a stock increases or decreases. For example, if inventory is expressed as a stock, production rate and shipment rate can be expressed as flows (Figure 3.1).

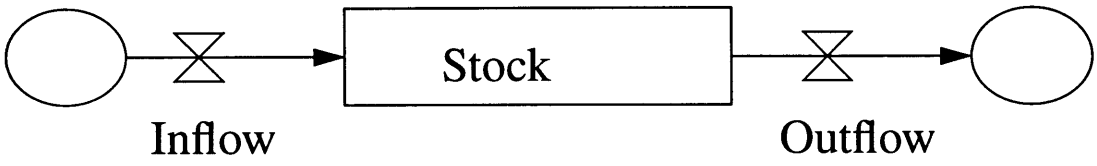


Figure 3.1: Stocks and Flows

Production rate increases inventory, and shipment rate decreases inventory. It is important to note that stocks depend only on flows and not other level variables. Similarly, flows are solely dependent on stocks and not other rates. Mathematically, stocks are the integration of flows (Figure 3.2).

$$stock = \int (inflow - outflow)dt$$

Figure 3.2: Stock and Flow Equation

3.3.2 Feedback loops

One of the most powerful aspects of system dynamics is its use of feedback loops. A feedback loop is a closed path within which all decisions are made. For example, someone will make a decision based on the observed state of the model. The decision will alter the parameters, chang-

ing the state of the model, and lead to even more decisions and changes. Basically, the feedback loop implies a circularity of cause and effect. There are two types of feedback loops - positive and negative.

3.3.2.1. Positive Feedback Loops

A positive feedback loop is one which activity changes the condition of the system in such a direction as to produce even greater activity. For this reason, positive feedback loops are often called "reinforcing loops." Positive feedback loops lead to exponential growth or decline. For example, sales force, number of orders, profit, and a manager's ability to hire more salespeople may form a reinforcing feedback loop (Figure 3.3). When the sales force increases, the number of orders also increase. The larger quantity of orders adds to the company's profits. The profits increase the managers ability and incentive to hire more salespeople. This leads to more sales people and even more orders. In this scenario, the sales force, number of orders, profits, and manager's ability to hire more salespeople all increase exponentially (Figure 3.4).

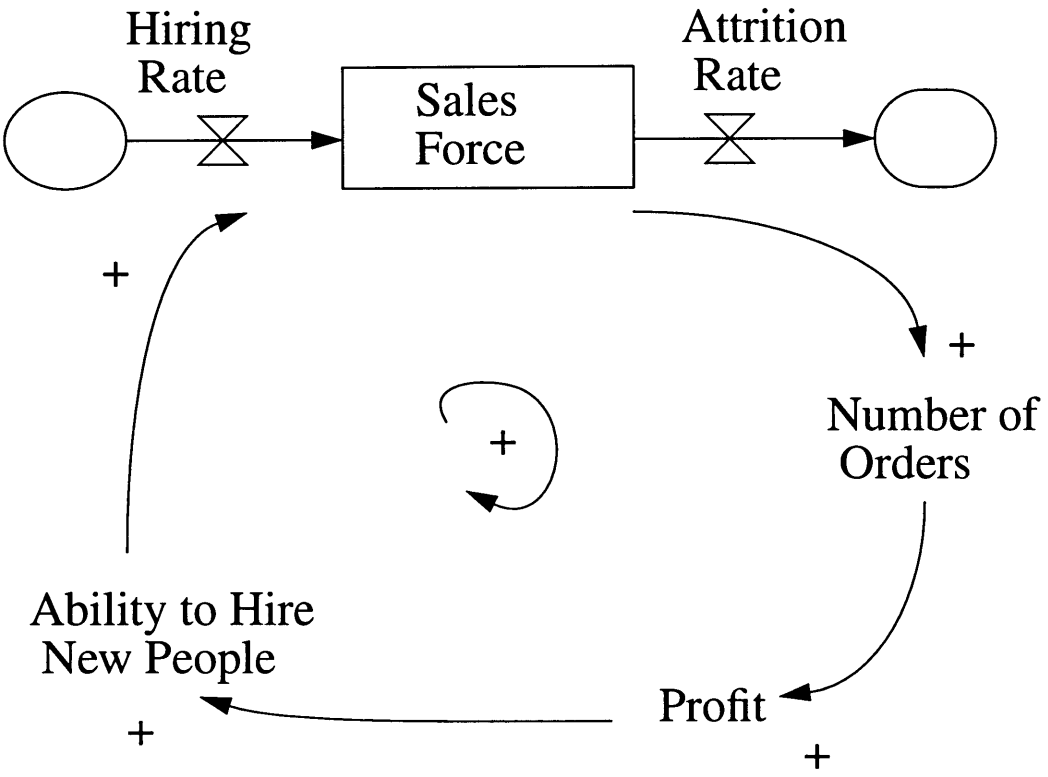


Figure 3.3: Salesforce Positive Feedback Loop

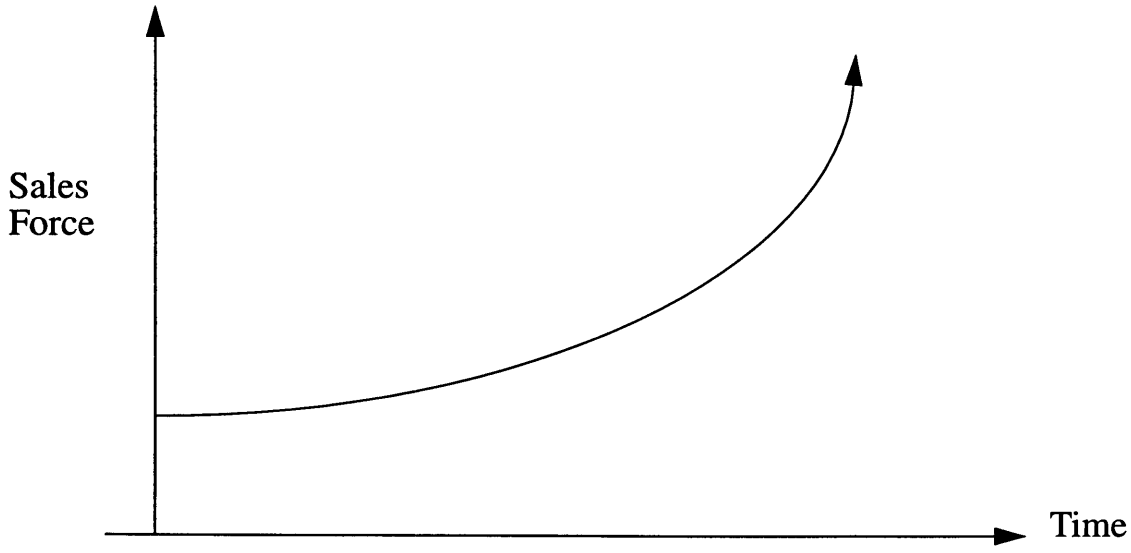


Figure 3.4: Graph for Positive Feedback Loops

There are many real life examples of positive feedback loops. One example involves firms entering an industry and choosing their locations so as to maximize profit. Assume that a company's profit increases if they are near other firms - who are typically their supplier or customers. The first firm which enters the industry chooses a location purely due to geographical preference. The second decides based on preferences including being located near the first company. The third company is influenced by the first two, and so on. The large number of high-tech companies in Santa Clara County California (Silicon Valley), is the result of such a positive feedback loop. [11] In the late 1940s and early 1950s, key people in the electronics industry - the Varian Brothers, William Hewlett and David Packard, William Shockley - chose to open companies near Stanford University. The local availability of engineers, supplies and components that these early companies helped furnish, made Santa Clara County extremely attractive to the 900 or so firms that followed [11] (Figure 3.5).

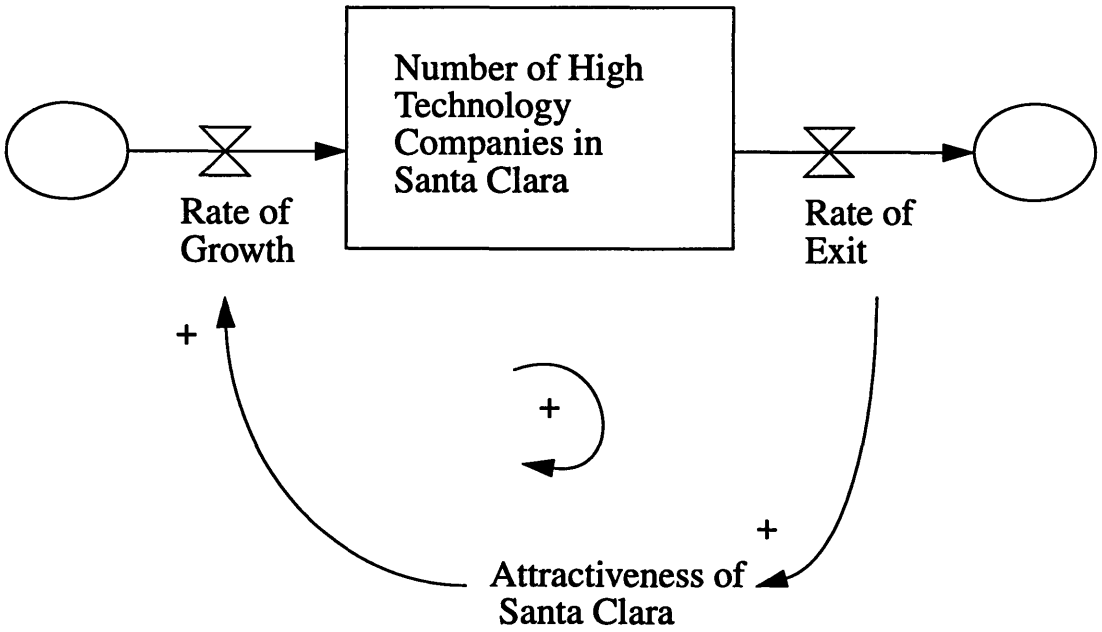


Figure 3.5: High Tech Companies Positive Feedback Loop

The history of VHS and Beta is another good example of positive feedback loops. The VCR market started with two competing formats selling at approximately the same price. Panasonic owned the VHS format, and Sony owned the Beta format. They both had equal market shares. Due to luck and corporate maneuvering, Panasonic's VHS format, accumulated a slight advantage. The larger number of VHS recorders encouraged video outlets to stock more prerecorded taped in VHS format. This, in turn, enhanced the value of owning a VHS recorder and even more people purchased the VHS format recorders. This positive feedback loop allowed the VHS customer base to grow exponentially (Figure 3.6). Within a few years, the VHS format had taken over virtually the entire VCR market [1]

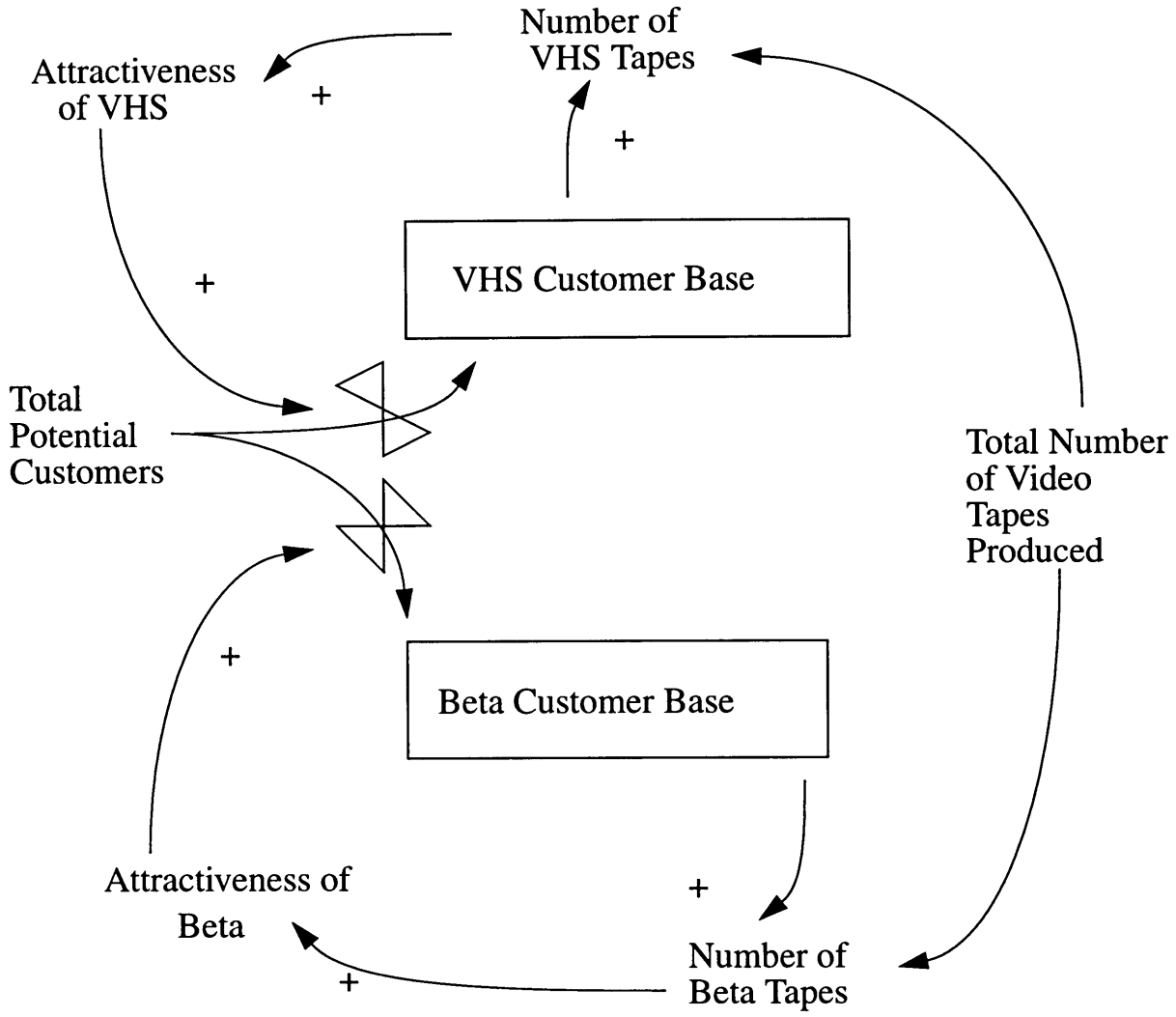


Figure 3.6: VHS vs. Beta System Dynamics Model

Nothing in the real world is completely made up of positive feedback loops. There will always be limits. In the VHS and Beta example, the limit is the customer base. Once VHS owned the entire customer base, it could no longer continue to grow exponentially. This means there exists another type of feedback loop - negative feedback loops.

3.3.2.2 Negative Feedback Loops

Negative feedback loops are goal seeking and adjust activity toward some target value - typically called the equilibrium. For this reason, negative feedback loops are often called "balancing feedback loops." For example, number of orders, shipment rate, invoice mistakes, and customer satisfaction may form a negative feedback loop (Figure 3.7). If the number of orders increase, the shipment rate also increases, assuming that there are no inventory limitations. An increase in shipment rate increases the percentage of mistakes in the invoice. More mistakes lead to reduced customer satisfaction, which decreases the rate of sales and number of orders. On the other hand, when the number of orders is low, rate of shipment is slow, reducing mistakes in the invoice. This increases customer satisfaction and hence the number of orders. The loop is balancing because it prevents the number of orders from being too high or too low. If the number of orders is too high, customer dissatisfaction drives it down. If the number of orders is too low, an increase in customer satisfaction drives it up (Figure 3.8).

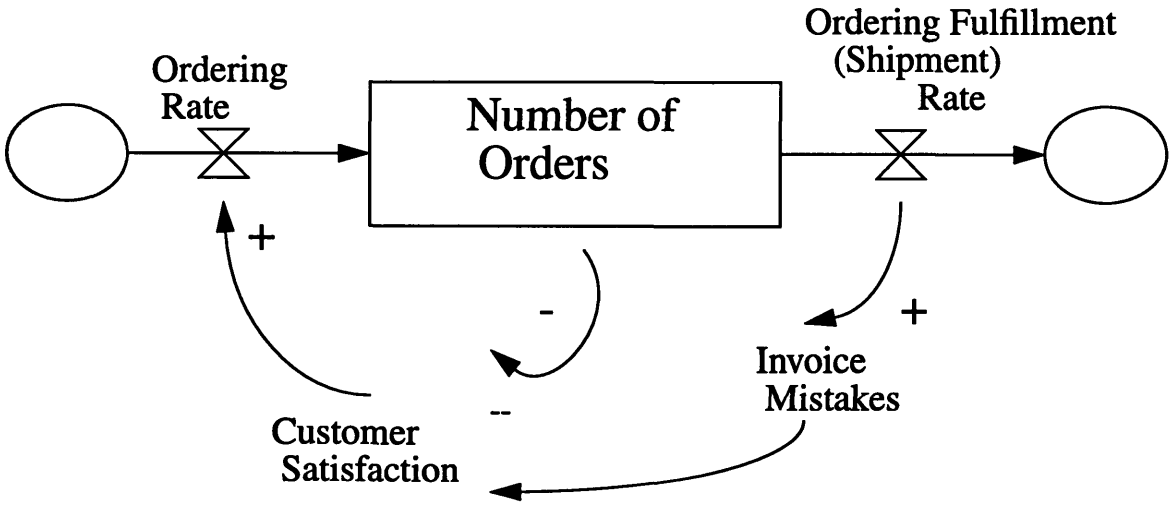


Figure 3.7: Number of Orders Negative Feedback Loop

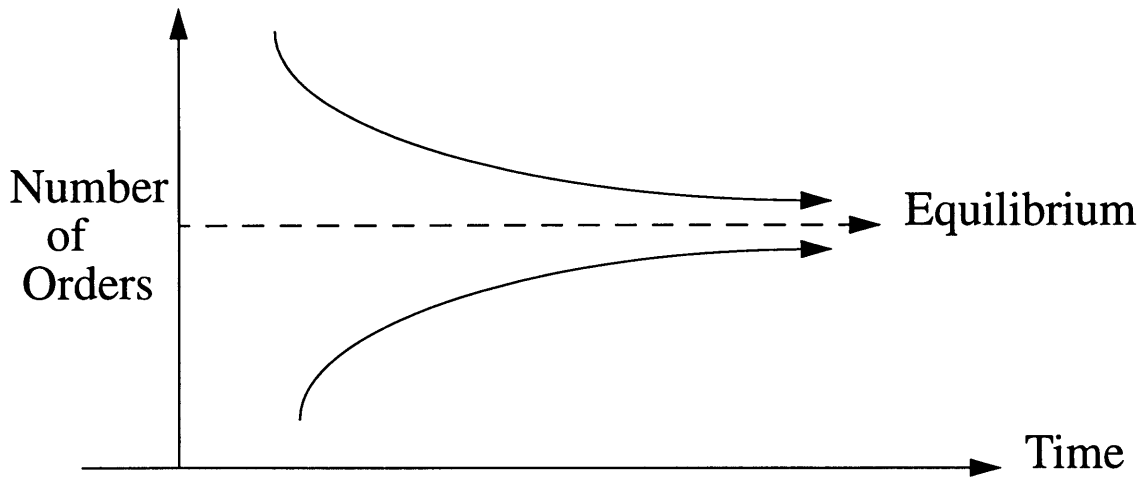


Figure 3.8: Graph for Negative Feedback Loops

There are also many real world examples involving negative feedback loops. One example involves a study done by Art Schneiderman, vice president of quality and productivity improvement at Analog Devices Inc. He concluded that, typically, defect level decreases at a constant rate so that when plotted on semilog paper against time, it falls on a straight line. In other words, if it took 3 months to find and solve the first 10 errors, it would take 3*2 = 6 months to find and fix the next 10 errors. This process could continue up to a certain limit, often due to inherent equipment limitations [12]. This phenomenon is due to the fact that the biggest and easiest to solve errors are typically solved first. Also, there are more errors in the beginning, so that it is easier to find errors. With each iteration, errors are harder to find and more difficult to solve. The following negative feedback loop models this phenomenon (Figure 3.9).

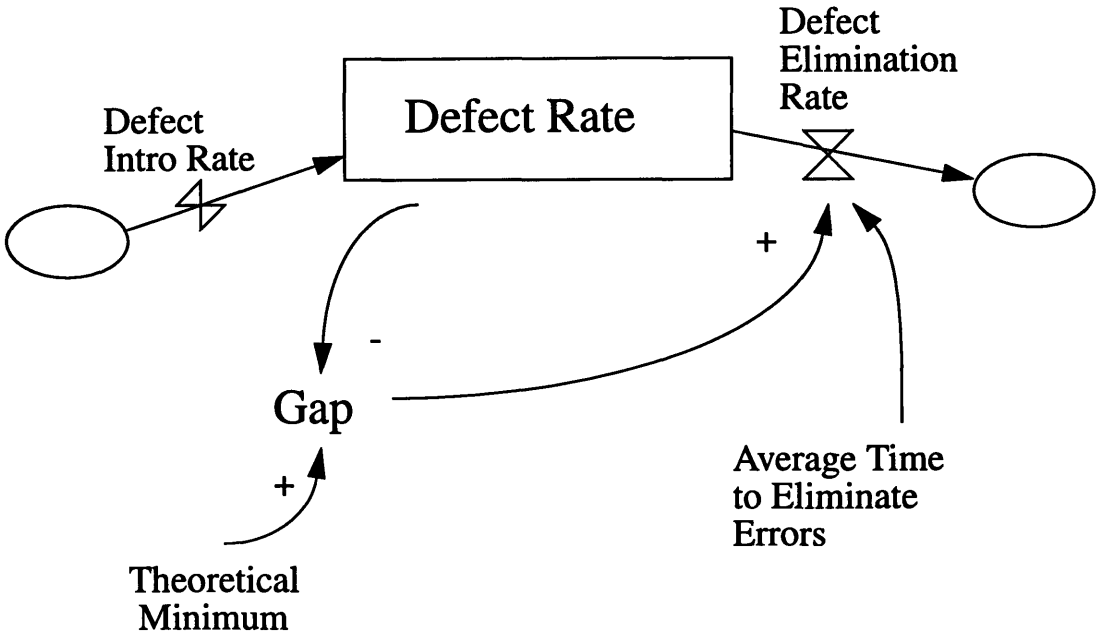


Figure 3.9: Defect Rate Negative Feedback Loop

3.3.2.3 Why Feedback Loops are of Interest

Feedback loops are of particular interest to this thesis project because they are one of the major strengths of using system dynamics modeling. Complex systems, such as those involving project management, are highly connected, and there is a high degree of feedback between sectors [9]. A model which does not incorporate feedback effects tends to rely on exogenous variables. In other words, the variables in the model are given by a set of numerical values over time. This is less realistic because variables in a model often effect one another significantly. Ignoring feedback can result in policies that are delayed and diluted, as well as decisions which generate unanticipated side effects [9].

One study conducted by John F. Collins, Professor of Urban Studies at MIT, and Jay Forrester showed that actions taken to alleviate the difficulties of a city can actually make the situation worse [13]. For example, the mayor of a town may increase the number of low-income housing to alleviate the problems of homelessness in the city. Thinking linearly, the solution is sound - low income housing will lead to fewer homeless people, increasing the attractiveness of the city. However, an unintended side effect may be that land used for projects takes away land which can be used to build companies and increase the number of jobs available. Also, the increase in cheap housing, will increase the attractiveness of the city to people with low incomes. As more people with low incomes move in, the demand for low income housing increases. Excess low income housing attracts more low income families. The positive feedback loop involved is far from insignificant, and causes a decline in the city's economic condition.

The use of feedback loops also implies that system dynamic models are non-linear. Often times, to simplify a simulation model, all the relationships in a system are assumed to be linear. Linearity, although mathematically convenient, is almost always unrealistic [9]. For example, a company may try to model a relationship between inventory and shipment. Collected data may show that when the inventory of products in a warehouse is 10% below normal, shipments are reduced by 2% because certain items are out of stock. If the relationship was linear, an empty warehouse will still allow 80% of the shipments to be made - obviously unrealistic. As another example of a non-linear system, consider the plight of the passenger pigeon. Before the colonization of North America, there was a large population of passenger pigeons. Since they caused extensive damage to crops, they were hunted both as a pest and for food. For years, hunting had relatively little impact on the birds, who were able to reproduce quickly. However, the fertility of the pigeons

depended nonlinearly on their population density. As hunting reduced the population, fertility fell, accelerating the decline in population. Lower population lowered the birth rate, and so forth in a positive feedback loop. By 1914, the passenger pigeon was extinct [9].

Studies have shown that people significantly and consistently underestimate exponential growth and decline, tending to extrapolate linearly rather than exponentially [4, 13]. System dynamics modeling incorporates the non-linearities and can help improve people's intuition on exponential growth and decline. Since non-linear relationships are more realistic, it makes sense to train employees using non-linear modeling techniques such as those in system dynamics.

3.3.3 Delays

System dynamics also takes into account the delays in a system. Delays in a system cause oscillations (Figure 3.10). Take for example, a manufacturing plant. If the demand for a product increases, inventory will start dropping. A manager may want to increase inventory by increasing the rate of production. However, since production is not instantaneous, inventory continues to drop. The manager will further increase his order. After a month or so, the first set of products he ordered is finished and added to inventory. The manager waits until inventory is high enough and stops increasing his orders. However, there are still large amounts of work in progress (WIP) due to his past orders, and he will end up with excess inventory. For another month, the manager may decrease his orders because inventory is too large due to the large number of WIP being completed. A month later, the decreased orders start coming out of WIP, and the manager is again met with insufficient inventory. This cycle of excess inventory and a depleted inventory continues despite the fact that customer demand has since remained constant.

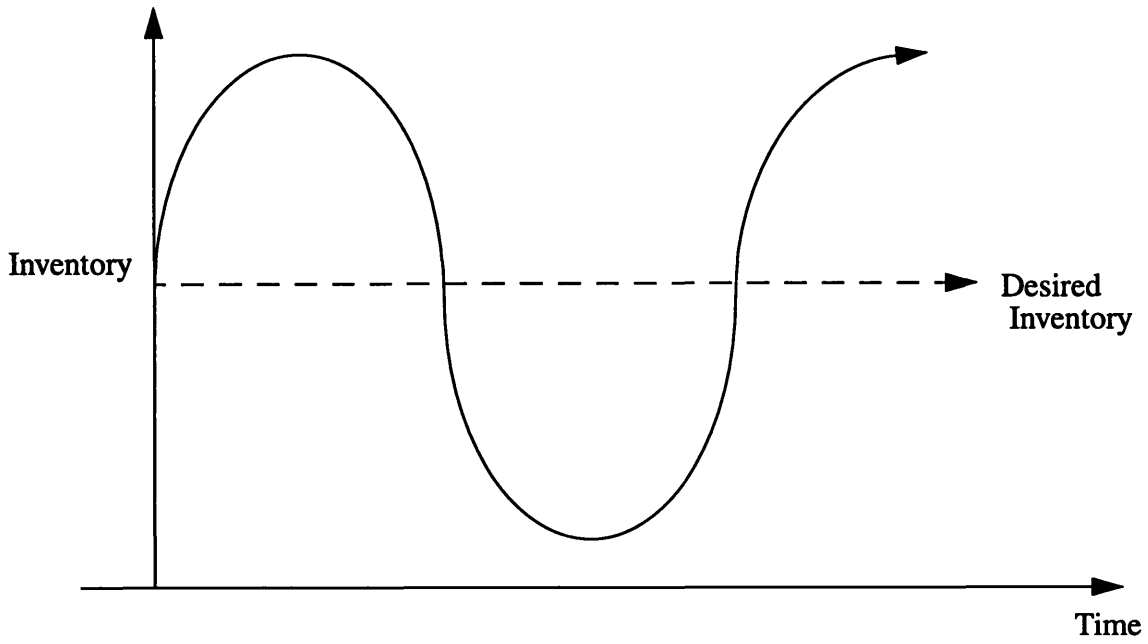


Figure 3.10: Graph for Delays Within a System

There are also real examples of delays and oscillation in the real world. One such example involves the building and profitability of oil tankers [14]. When oil tankers were making a large profit, investors were encouraged to build more oil tankers. It took approximately three years for an oil tanker to be ordered and built - a delay of three years. During those three years, the few existing oil tankers were making large profits. Then three years later, oil tankers started coming out of production. There were too many oil tankers, profitability per tanker dropped, and some tankers were even sold very cheaply for scrap metal. Investors stopped the building of tankers - after all, it didn't make sense to build something that would be unprofitable. Another three years later, there was once again a shortage of oil tankers since building of tankers had basically ceased. The existing tankers made large profits, and again many investors jumped into the ship building business. The cycle continues periodically regardless of economic conditions.

System dynamics modeling uses delays to make simulations more realistic. Most useful system dynamics models are made out of both negative and positive feedback loops with some delays.

4. Using System Dynamics in Project Management Simulations

4.1 Why Model Project Management?

There is a need to simulate project management because project management is very complex and often poorly understood. Cost overruns are the rule rather than the exception [15]. Projects are often delayed to the point where the market conditions for which they were designed have changed [15] Client specifications change, often disrupting the entire organization. Many managers have misconceptions about adding new employees to a project to increase productivity [16] This section of the paper will describe common responsibilities of a project manager, as well as some typical pitfalls in the project management process. This section will also explain why system dynamics is an efficient and realistic way of modeling project management.

4.1.1 Project Management Responsibilities

A project manager assumes many responsibilities. Project managers must estimate a reasonable budget, schedule, and headcount for a project. They are often in charge of allocating project resources, such as manpower, facilities, and equipment. They decide who to hire, who to train, and how to motivate employees to get the maximum effective work week. There are many risks such as client scope changes, unreliable subcontractors, and attrition of experts on the team which

a good project manager must be prepared for. Furthermore, project managers are responsible for making crucial decisions when a project which falls behind schedule, is over budget, or has quality problems. For example, if a project is behind schedule, a project manager can increase headcount, pressure current employees to work overtime, revise the completion date, or lower the quality of the product. Each of these choices come with its own ramifications, which must be weighed by the project manager.

The above paragraph only mentions the most basic responsibilities of a project manager, and already it is complex. Many of the variables interact. For example, hiring affects parameters such as the budget, productivity, and allocation of resources. Overtime can cause fatigue affecting the quality of the product and increasing attrition rate. The human mind does not have the capability to make optimal decisions faced with such a complex, nonlinear, interactive system [15]. Even the best project manager is bounded by limitations of attention, memory, and information processing capability. For these reasons, many managerial misconceptions exist.

4.1.2. Examples of Common Management Misconceptions.

In this section three common managerial misconceptions are discussed: Brook's Law, the 90% Syndrome, and Overtime/Schedule Pressure. Brook's Law states that "adding more manpower to a late software project makes it later" [16]. A dangerous misconception is that humans and time are interchangeable. In other words, the more people on a project, the less time it takes and vice versa [16, 18]. This hypothesis does not take into account the communication overhead involved when many people work on one project. Furthermore, when a task cannot be partitioned because

of sequential constraints, the application of more effort will obviously not speed up the project at all [16]. Increasing headcount also means adding people who are unfamiliar with the project. Each new worker must be trained on the technology, the goals of the effort, the overall strategy, and the plan of work [17]. The experienced employees often become responsible for bringing the new hires up to speed. This takes away time that they could be using to work on the project. A good project manager must take these issues into account and resist the temptation to add more people at the end of a project in order to make a deadline.

Many projects also suffer from the 90% syndrome, where a project is thought to be 90% finished for half of the total time required. This is partially because people on the project are often overly optimistic [16, 18]. For example, software engineers may assume that no major design errors will crop up during integration testing. Workers may also purposely distort the status of the project to the manager [18]. A good project manager must realize this phenomenon, monitor progress more carefully, and use better progress estimation techniques.

The last example includes overtime and schedule pressure. When a project falls behind schedule, there is temptation to push for overtime. It seems intuitive that longer hours lead to higher productivity. This is often not the case. Overtime does not increase productivity in the long run [17, 18]. When employees put in overtime, their productivity will temporarily rise. However, if they work a significant amount of overtime, burnout lowers their productivity, increases the error rate, and decreases the error discovery rate [17]. Studies have shown that work force turnover increases when schedule pressures persist in an organization [18]. The company typically loses its best employees, who can easily find work elsewhere. Their replacements are new hires who are unfa-

miliar with the project and need extra training. The net result is often a project which is even further behind schedule.

4.2 Why Use System Dynamics?

Because project management is so complex and poorly understood, it is important to train future project managers on business simulation games. This section explains why system dynamics should be used as the underlying model in such games.

Large scale projects belong to the class of complex dynamic systems. Such systems consist of multiple interdependent components, are highly dynamic, involve multiple feedback processes, and involve non-linear relationships [15]. Each of these aspects of project management are addressed by system dynamics modeling.

Project managers make many decisions involving interdependent variables. For example, subcontracting a piece of the project effects budget, headcount, and risk. Multiple interdependencies complicate analysis beyond the capabilities of mental models [15]. However, system dynamics models represent multiple interdependencies very well. In fact, one of the major uses for system dynamics is to model interdependencies so that causal impacts of changes can be easily traced throughout the system [15].

The project management position is also a very dynamic position. Processes such as training unfold over time and never happen instantaneously. There are many delays in the process caused

by hiring, responding to quality issues, and dealing with sudden, unexpected changes in project scope. For example, after hiring a new employee, there is a large delay in training the person on skills required for the specific project. During this time, many of the experienced hires may be responsible for bringing the new person up to speed. As a result, productivity decreases temporarily. System dynamics was developed to handle exactly such dynamics [15]. Of all the formal modeling techniques, system dynamics has the most highly involved guidelines for proper analysis, representation, and explanation of the dynamics of complicated managerial systems [15].

There are many feedback loops involved in project management. For example, when a project falls behind schedule, a manager may pressure employees to work overtime. The overtime may bring the project back on schedule. This is an example of a negative feedback loop. However, if the project is very behind schedule and employees must put in weeks of overtime, fatigue may set in causing quality problems and low employee morale. This may cause the project to fall behind schedule even more. This is an example of a positive feedback loop. System dynamics is the most efficient and effective way of modeling when there are many feedback loops [15].

Project managers must handle nonlinear relationships between many variables. In complex systems, causes and effects rarely have simple, proportional relationships [15]. One example is testing a large project which includes many interacting components. Assuming integration and modular testing, a project with three components takes more than three times longer to test than a project with only one component. System dynamics, more than any other formal modeling technique, stresses the importance of including nonlinearities in formulating a model [15].

Given the strengths of system dynamics, system dynamics modeling lends itself to project management very naturally. Playing a simulation game with a system dynamics engine can help trainees avoid common pitfalls of project management. Trainees can test their mental model and discover their own misconceptions.

5. The Existing ACE Project Management Fundamentals Training

Before delving into a detailed description of the changes made to the current Andersen Project Management Fundamentals training game, it is first important to understand the existing method of training. The goals of the training as well as each step will be explained in this section.
The purpose of ACE Project Management Fundamentals is to train newly promoted project managers. The ultimate goal is for these employees to become effective project managers. The current game consists of a number of sections: pre-reads, project initiation, and six case studies. Each of these sections will be described briefly. A complete, detailed explanation can be obtained from ACE.

Prior to attending the training program, students are expected to prepare by reading relevant articles on project management. These articles are provided by ACE, and include topics such as the importance of project management and specific management concepts. Furthermore, students are expected to be familiar with acronyms such as BCWS (Budgeted Cost of Work Schedule), the sum of all budgets for work scheduled to produce program deliverables.

5.1 Orientation and Initiation

The formal training program begins with project orientation and initiation. Coaches, who are ACE volunteers, greet the trainees and give them an overview of the course. The coaches are responsible for explaining key project management concepts, the project management process, and the role of the project manager. The acronym SQERT is introduced during this initiation phase. SQERT stands for Scope, Quality, Effort, Risk, and Time - five major components of project management. Next, the students are given their specific project to manage, the "Order Capture Project." A program manager, an ACE volunteer, explains the process of confirming the baseline. Confirming the baseline involves using estimation techniques to validate a proposed budget and schedule. Trainees are reminded that project managers have the opportunity to negotiate the baseline.

5.2 Case Study I

The first case involves confirming the baseline.The trainees are responsible for analyzing the program-level Order Capture Project documentations and negotiating a more reasonable baseline with the program manager. In order to do this, trainees must understand program-level documentation as it relates to the project and have a solid understanding of the expected project deliverables. They must also verify the reasonableness of the project baseline regarding scope, quality, effort, risk, and time. As it turns out, the baseline presented by the program manager is not feasible. For example, more training time than planned will be needed, and there are no full-time users on the project team. The trainees must defend their request for increased time and budget. The goal of this section is for the project manager and program manager to reach a consensus on the

baseline. However, regardless of any baseline changes agreed upon by the project manager and program manager, the training course continues using the original, unreasonable baseline.

5.3 Case Study II

The second case focuses on developing the project plan with emphasis on scope, effort, and time. Participants review and analyze the initial project plans for completeness, correctness, and reasonableness. They are then to develop a list of observations, concerns, and suggested revisions to the initial project plan. Finally, the trainees meet with the program manager to propose and justify suggested revisions to the initial, suboptimal project plan. For example, a player may point out that project milestones are set too far apart or that the basis used for estimating work are not documented. Trainees are reminded that project managers can make a change to project plans as long as it does not effect the overall budget, schedule, quality, and dependencies of the project. However, none of the suggested changes will effect the next case. The project will continue using the original project plan despite the improvements proposed by the students.

5.4 Case Study III

The third case also involves developing the project plan, but this time, with emphasis on quality and risk. ACE students are asked to identify and prioritize risks and develop mitigation strategies for risks that apply at the project level. Some risks identified by trainees may include expanding solutions requirements, non-availability of project personnel, and underestimated efforts. Trainees

recommend ways to reduce risk while maintaining a high level of product quality. Again, these recommendations are ignored in the next case.

Included in the third case is a Project Status Assessment and Measurement Workshop. This workshop teaches ACE students how to measure the progress and future headcount, budget, and scheduling needs of the project.

5.5 Case Study IV and Case Study V

The final two cases involves assessing project status and giving presentations. The students are required to assess project status using information from project reports, identify corrective actions, and prepare a report. They must then present the project status to both the program manager and the program executive. Trainees are expected to gear their presentations appropriately to the different audiences. The status report to their project manager should be somewhat technical and more detailed. The presentation to the executive manager, on the other hand, should be higher level and less technical. Trainees are evaluated on the accuracy, completeness, and professionalism of their report.

The training session ends with a wrap-up given by the coach. The coach recaps the skills developed during the training, and reminds students to apply what they have learned to their actual job assignments.

6. The Project Management Game Using System Dynamics

One major drawback to the original game is its lack of dynamics. Any suggestions from the students, regarding the parameters, cannot be incorporated into the game. In order to improve the game, ACE has decided to look into the possibility of a using a system dynamics model as its game engine. The engine would allow the trainees to input major parameters. For example, the trainees may decide on the percentage of the development process they want to spend on testing. Creating a system dynamics model from scratch is very time consuming. The equations relating the variables need to be well tested. Feedback loops need to be analyzed to make sure they output realistic numbers. Furthermore, data generated by the model should be tested against actual data from real case studies. In project management, there are many soft variables such as team morale and level of fatigue. Data used for equations involving these variables are often obtained through extensive interviewing of company employees.

Given that creating a realistic system dynamics model from scratch is very difficult and time consuming, it was decided that the initial engine should be based on an existing model. The model chosen was one created by Tarek Abdel-Hamid for his 1984 Ph.D. thesis from the MIT Sloan School of Management. His thesis included a well-tested and accepted system dynamics model of the software project management process.

The existing model by Abdel-Hamid was created using DYNAMO, an old and outdated system dynamics modeling software package. DYNAMO is not compatible with many of the newer modeling software tools. Since putting the game engine on the ACE intranet is a possibility for the

future, it was important that the model be created using a newer software package which may include networking and web capabilities in the near future. The model was, therefore, ported to Vensim, a more current software package which is compatible with many other modeling tools on the market. This Vensim model serves as an example of how system dynamics can be used to successfully model project management. It will also be used as the base model for the ACE game engine. ACE's needs for the game engine is obviously not perfectly replicated by Abdel-Hamid's model. There will be changes to both the structure and parameters to the base model. The initial design changes suggested by ACE will be addressed in this section of the paper.

6.1 Abdel-Hamid's model

Before delving into the design changes proposed by ACE, it is first important to understand the overall structure of the original software project management model from Tarek Abdel-Hamid's thesis. The model is fairly large, consisting of five major subsystems and approximately 200 equations. The ported Vensim equations are included in Appendix A.

The overall structure of the model is shown in Figure 6.1. The various sections influence each other as shown by the arrows. For example, the development section generates how many tasks are completed. The controlling sector uses the number of completed tasks to decide if more effort and people are needed. If more people are needed, the human resource sector makes decisions on hiring new employees. A complete system dynamics model of these and other subsectors is included in Appendix B.

Some of the major inputs of this system dynamics model are estimated man-days needed, estimated size of project, initial work force size, and percent of effort for development and testing. The major outputs include scheduled completion date, equivalent work force, cumulative errors committed, and cumulative man-days expended. Each of the subsystems will be reviewed very briefly in these next few sections of the paper. For a more detailed description, as well as the actual DYNAMO equations, please refer to "Software Project Dynamics. An Integrated Approach" by Tarek Abdel-Hamid and Stuart E. Madnick [18].

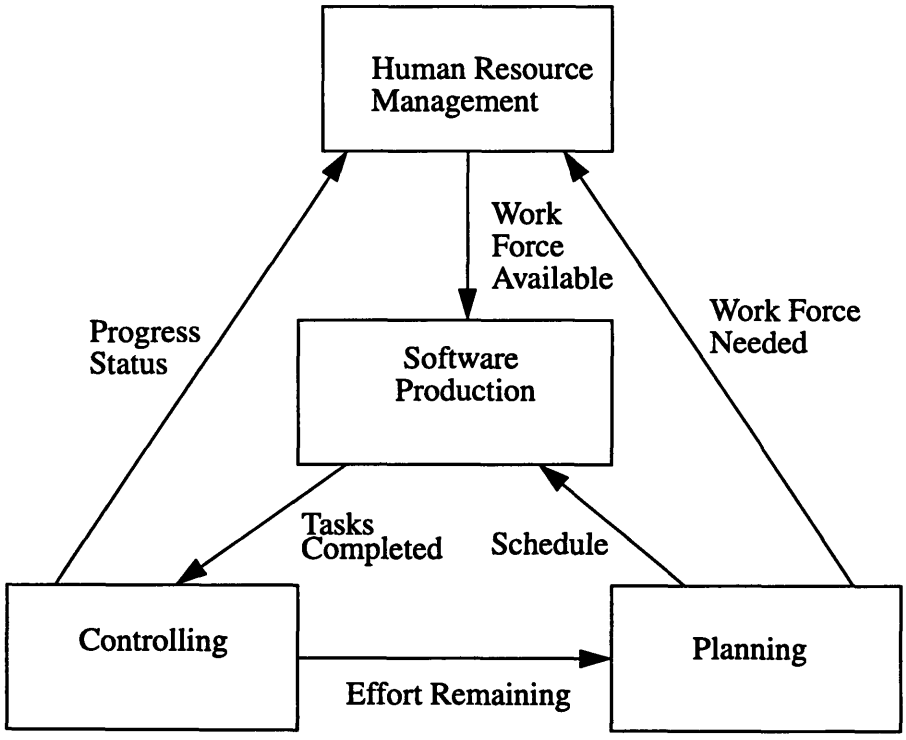


Figure 6.1: Abdel-Hamid Model [18]

6.1.1 Human Resources

The human resource sector of the model controls the number of people working on the project. There are two main types of people working on the project: experienced employees and new hires. New hires assimilate into experienced workers through training overhead. Hiring rate increases when there is a larger discrepancy between the number of people needed and the actual number of employees.

6.1.2 Software Production

The software production sector is made up of four subsectors: manpower allocation, software development, quality assurance and rework, and system testing. The manpower allocation sector handles what percentage of the manpower goes toward the various parts of the development process. For example, variables include the total cumulative man-days spent on quality assurance, rework, and training. The software development section calculates the amount of work done. This section takes into account the effects of overtime, exhaustion, and communication overhead to the overall project productivity. The quality assurance sector includes equations relating the error density, the error detection rate, and the amount of manpower needed to detect and fix an error. The error generation rate increases with schedule pressure, since there is often an inverse relationship between working faster and working better. This is because emphasis is placed on getting as many tasks done as quickly as possible. Less time is spent on trying to meet the quality objectives and testing for defects. The last major part of the software production sector is system testing. Testing can catch two types of errors - passive and active. Active errors are errors which lead to more errors. For example, an error in design is an active error since it will lead to errors in coding

and perhaps other aspects of lower level design. Passive errors, on the other hand, are errors which do not lead to more errors. Obviously, since active errors induce even more errors, they are of greater concern. The system testing sector includes equations which calculate the percentage of passive and active errors, the cumulative tasks tested, and the error correction rates.

6.1.3 Control

The controlling subsystem of Abdel-Hamid's model includes measurement, evaluation, and communication. This section first measures an activity such as total man-days still needed. It then evaluates its significance by comparing information on what is actually happening with some standard or expectation of what should be happening. For example, total man-days still needed would be compared to total man-days perceived still needed. Next, a report of what has been measured and assessed is communicated, so that the behavior can be altered if the need for doing so is indicated. As an example, if total man-days needed is much higher than the total man-days remaining, hiring rate may increase.

6.1.4 Planning

In the planning subsystem, initial project estimates are made at the start of the project, and then the estimates are revised when necessary throughout the project lifecycle. For example, there exists an initial scheduled completion date. If the project falls behind, the scheduled completion date will need to be pushed back.

6.2 Porting the Model from DYNAMO to Vensim

The original Abdel-Hamid system dynamics model was created using DYNAMO, an old system dynamics modeling software package. It was necessary to port the model to Vensim because DYNAMO lacked many of the features new modeling software packages contained. Furthermore, DYNAMO is not compatible with the newer modeling tools on the current market.

As a whole, porting the Abdel-Hamid model from DYNAMO to Vensim was successful. To test the overall correctness of the ported model, three steps were taken. First, the equations were checked by hand with the published equations in "Software Project Dynamics - An Integrated Approach" [18], as well as the original equations in Tarek Abdel-Hamid's Ph.D. thesis. Next, variables were checked for obvious errors. For example, work force was tested to make sure the number of employees was never a negative number. The number of tasks tested was compared against the number of tasks developed to ensure that the number of tasks tested never exceeded the number of tasks developed. Anything cumulative (Cumulative Man-Days, Cumulative Tasks Tested, Cumulative Training Days, etc.) was graphed to show that it was monotonically increasing. The final test on the ported Vensim model was to compare the results it generated to the ones generated by the DYNAMO model. "Software Project Dynamics - An Integrated Approach" included DYNAMO generated results from an example software project with fictitious parameters as well as an actual case study on a NASA software project. The ported model ran for 400 simulated days, with time steps of half a day. For most variables, the results matched the DYNAMO model simulations exactly. A few variables, however, had discrepancies starting at about day 350. This discrepancy was analyzed, and it seems as if the problem is caused by an equation from the testing subsector. This was deducted because testing does not occur until the very end of the project at approximately the same time as when the discrepancies started to occur. Equations

regarding the testing subsector were analyzed. One equation regarding the stock Planned Testing Size in Man-Days did not seem correct. The DYNAMO equation states:

Planned Testing Size in Man-Days = Planned Testing Size in Man-Days + Time Step * Rate of Increase in Development Man-Days Due to Discovered Tasks + (Rate of Adjusting the Job Size in Man-Days if the Fraction of Effort for Testing is greater than .9).

In other words, if the Fraction of Effort for Testing is greater than .9, Planned Testing Size in Man-Days is the integral of Rate of Increase in Development Man-Days Due to Discovered Tasks plus the integral of Rate of Adjusting the Job Size in Man-Days multiplied by Time Step. It does not make sense to multiply Rate of Adjusting the Job Size in Man-Days by the Time Step because it seems unreasonable for the rate to depend on the time step chosen for a specific simulation run. Hence, it is speculated that there is a missing parenthesis in the original equation, which should read:

Planned Testing Size in Man-Days = Planned Testing Size in Man-Days + Time Step * (Rate of Increase in Development Man-Days Due to Discovered Tasks + (Rate of Adjusting the Job Size in Man-Days if the Fraction of Effort for Testing is greater than .9)).

An e-mail requesting a discussion on this potentially erroneous equation was sent to Dr. Tarek Abdel-Hamid in April, 1997. As of May, 1997, no response has been received.

6.3 The Abdel-Hamid Model and ACE Specifications

Although Tarek Abdel-Hamid's model of software project management is solid and well-accepted, it does not meet all of ACE's requirements. Some parts are too detailed for ACE's purposes, and there are some aspects of project management which ACE would like to place more emphasis upon. ACE calls their game engine the PERIOD1 engine, and has sent initial design specifications regarding changes to Abdel-Hamid's model. The design specification are included in Appendix C of this paper. The next section of the paper is a response to the ACE PERIOD1 engine specifications.

A few of the initial ACE designs took into account very detailed variables, such as an individual team member's skill level. Before delving into the details of the ACE design specifications, it is first important to know what types of decisions are most effectively modeled using system dynamics. System dynamics is a method of modeling which is good at showing changes and trends in a system. The strengths of system dynamics modeling lie in representing higher level system structures rather than depicting lower level details.

It is not effective to include too many micro decisions in a system dynamics model because minor decisions can make the simulation more complex and even misleading. For example, assume there are two skill levels - inexperienced and experienced. A student can then decide to test how training effects the overall productivity of a team. It would be easy to observe, via Vensim graphs and tables, that training improves the productivity of new hires by 50%, but experienced workers by only 5%. Policies such as training new hires more aggressively can then be implemented to maximize team productivity. If the model had been more detailed, and each team member was given a

specific personality and skill level, the fact that new hires benefit more from training may be lost. It might be mistaken that certain personalities learn faster or that a specific employment background makes someone more productive.

This is not to say that certain human aspects - emotions, personality, tolerance - do not factor into the equations. They do. However, the pros and cons of a more detailed model should be weighed carefully.

6.3.1 Calculating Base Efficiency

In this section, ACE's calculations for base efficiency will be addressed. The ACE specifications gives each team member an efficiency level (skill level of person/skill level of task). This may be too detailed to effectively use system dynamics modeling. In the Abdel-Hamid model, base efficiency is calculated using three major variables. Experienced workers receive an efficiency parameter of 1. Newly hired work force receive a parameter of .5, meaning that they are only half as efficient as experienced workers. Base efficiency also depends on the average amount of time the team members can devote to a specific project. For example, a .5 means the average team member spends half his time on this project and the other half on other projects. Finally, we give each person a parameter of .6, meaning that under normal conditions, about 60% of a person's work day is devoted to the project. The other 40% takes into account coffee breaks, personal conversations with office mates, and other ineffective time usage.

Individual task dependencies are not dealt with in the Abdel-Hamid model. However, it does take into account the fact that there are multiple phases of completing a task - development, testing, and quality assurance, which depend on one another. For example, tasks cannot be tested before being developed.

6.3.2 Calculating Changes to Efficiency

There are a few variables which can increase and decrease efficiency. ACE's suggestions to these variables, as well as variables in the Abdel-Hamid model, which are not mentioned by ACE, will be addressed in this section.

6.3.2.1 Communication

ACE suggests calculating communication efficiency using the number of communications used for actions divided by the number of actions requiring communication. For example, employees may have communicated ten times during the course of a task, where communication can be measured by the number of meetings held, telephone calls made, e-mails sent and so forth. The task, however, may be set by ACE to require employees to communicate twenty times. In this case, communication efficiency would be 10 /20 = .5, half of the optimal communication efficiency. In the Abdel-Hamid model, the amount of communication a project needs depends on how many people are working on the project. The more people there are on a team, the more communication is needed to coordinate the project. The Abdel-Hamid method of calculating communication overhead is less detailed and easier to implement since there already exists well-tested data on the relationship between the amount of communication needed and the number of people working on

a project. However, ACE may prefer their level of detail, which can be incorporated as a multiplier to the Abdel-Hamid's communication overhead variable.

6.3.2.2 Overtime

The issue of overtime is a major component in the Abdel-Hamid model. Overtime is used whenever the time remaining is insufficient to finish the tasks still left. Team members do not chose to work overtime if it is not needed. If team members are working too many continuous days of overtime, they will have an exhaustion breakdown. During this time, they are no longer willing to work overtime. This current overtime subsystem can be improved. For example, working overtime has been shown to cause decreases in productivity as employees become more fatigued and frustrated. Also, too much overtime may cause morale to drop. ACE's suggestion to include self-induced overtime to the model structure is also a good one since employees may have incentives other than schedule pressure to work overtime. For example, employees may want to work extra hours for overtime pay or to position themselves for a promotion.

6.3.2.3 Development Need

The ACE specifications include a variable which matches an employee with his development need. For example, an employee may want to learn certain skills and hence prefer to be assigned to a specific task. The Abdel-Hamid model does not have an equivalent to this variable. Matching an individual with a specific task he likes, dislikes, or feel indifferent towards is too detailed for a project management system dynamics model. A better approach may be to incorporate a comparison between the average skill level needed for the project and the average skill level of the work force. This comparison can then be used to effect the overall team morale. If someone is working

on something they find too easy and tedious, their morale would decrease. On the other hand, if the job is too challenging, morale may also drop because the person becomes frustrated.

6.3.2.4 Team Morale

In the Abdel-Hamid model, morale is never addressed explicitly. There is a variable called "multiplier to productivity due to communication and motivational losses," which tries to capture the fact that as team size increases, motivation for working harder decreases. Morale is another area which should be expanded upon for ACE's training needs. Morale can have a very big impact on productivity. For example, morale may go down if the percentage of time spent on training falls below a certain threshold. The assumption would be that employees feel frustrated that their company is not investing in them. In addition, if morale is very low, attrition rate should increase.

6.3.2.5 Skill Match

The ACE design specifications include a variable called skill match. This variable compares the skill of an employee and the skill level needed for the task assigned to him. The Abdel-Hamid model is not detailed enough to match an individual to a specific task. Skill level is only taken into account by assuming more experienced workers are more efficient.

6.3.2.6 Work Force Mix

The ACE specifications do not mention work force mix, whereas the Abdel-Hamid model does. Work force mix is the fraction of experienced work force which influences the number of errors made per task. The rationale behind this is that new hires are more likely to make mistakes. There-

fore, too many new people on a task may decrease the overall productivity by introducing errors in the product which must be reworked at a later date.

6.3.2.7 Schedule Pressure

The ACE specifications also did not mention schedule pressure. Schedule pressure also influences the number of errors made. In the Abdel-Hamid model, schedule pressure influences the percentage of day the team members put into the project. It seems intuitive that this will increase productivity. In fact, it initially does. However, schedule pressure also increases the number of errors made per task. Working faster is not equivalent to working better. Quality assurance typically slackens during time of schedule pressure. The result is an increase in errors and lower quality work which may need to be redone at a later date. Therefore, net productivity may decrease due to schedule pressure.

6.3.2.8 Learning

Another factor noted in the Abdel-Hamid model, but not discussed by the ACE specifications is the age of the project. As time passes, productivity increases since employees become more familiar with the project. All team members, experienced or not, need to get acquainted with the dynamics of a specific project. The longer the project has been under production, the less uncertainty there is and the more standards there are. This increases productivity.

6.3.3 Scope Changes and Rework

The ACE specifications mention the importance of scope changes. Scope can easily be increased or decreased in the Abdel-Hamid model by changing a variable called "real job size in tasks." This should solve the scope creep issues addressed by ACE.

The ACE design specifications does not include rework as part of the PERIOD1 engine. Rework should definitely be included in the engine model because it has such a big impact on projects. Many times, rework can be a large factor in dragging a project beyond the scheduled completion date. For example, working people too hard may lead to more rework later, when the tasks are tested and errors are detected. This is a fairly important feedback loop which would be lost without considering rework.

6.3.4 Design Steps Recommendation

Given that there are a few discrepancies between the ACE specifications and the current software project management model, a few steps should be taken. The goal is to have a model of the PERIOD1 engine based on Abdel-Hamid's system dynamics model, but with appropriate changes as needed for ACE training purposes. Three major steps should be taken.

First, it must be decided which parts of the current model are too detailed or not relevant for ACE training purposes. It must be warned, however, that the original model has been well tested and the data it generates matches the results of many case studies. Any changes should be tested to maintain accuracy.

Next, additions to the base model must be incorporated. The specifications given by ACE includes a few variables and concepts which were not addressed in the Abdel-Hamid model. One example is treating tasks individually such as having certain tasks dependent on others and deducting efficiency when a team member starts on an task which is already underway. These issues can be incorporated by changing the structure of the current model slightly. Structural changes are obviously more difficult than variable changes and require more complete testing.

Finally, members of this project must agree on which parameters should be hard coded and which parameters should be set by the trainees. For example, it is possible that an Andersen Consultant does not take as many breaks as a typical software engineer. In that case, the percent of day an average consultant spends on projects may be 80% rather than 60%. Also, if Andersen only hires new consultants from other firms, the average efficiency for a new hire may be higher than the default .5 for an inexperienced hire. There are many parameters which can be preset based on research done on Andersen Consulting. Other variables should be set by the trainee. For example, trainees may decide on what percentage of the time he wants to spend on development versus testing. Trainees may also decide how many new people to hire. One issue to note again is that the current data sets used in the model are researched and tested. It is highly recommended to test new data sets before incorporating them into the model in order to maintain correctness.

7. Conclusions

7.1. The Final Presentation

This thesis project was wrapped up with a presentation in St. Charles, Illinois. The presentation was given by the author, and two undergraduate students, Eugene Fung and Autumn Steuckrath. About fifteen ACE professionals were in attendance, and the talk was video taped for ACE employees who had interest in the thesis project, but were not able to attend. Background research, goals obtained, and future milestones were mentioned in the presentation. Autumn Steuckrath presented the annotated bibliography, which is now on the World Wide Web for public access. The web page was designed and created by Steuckrath. Next, Eugene Fung recommended nine business simulation games for ACE to look into. Some of the games were commercial, while others could only be obtained through company contacts. All of the games contained sophisticated engines, and many had impressive user interfaces. Finally, the author discussed the Abdel-Hamid software project management model and addressed the ACE specifications regarding the game engine model. Emphasis was placed on using the Abdel-Hamid model as a base model and making changes, rather than start an entire model from scratch. The talk was well received by the ACE professionals, who asked questions throughout the presentation and joined in discussions. There were many ACE professionals who had heard of system dynamics modeling before and were eager to both increase their knowledge and put their knowledge to use in the creation of a business simulation game.

7.2. Project Plan for the Next Two Years

The overall goal for the next two years is to complete the PERIOD1 engine. On MIT's side, the project should be split into two areas of expertise. One area should be headed by Professor John Sterman, who is a professor in System Dynamics. His group should include one or two Ph.D. candidates with an interest in modeling project management using system dynamics. Next academic year (1997-1998), Professor Sterman will be on sabbatical. During this time, the chosen Ph.D. candidate(s) should take both the introductory system dynamics class and the advanced class at the MIT Sloan School of Management. By the time Professor Sterman returns from his sabbatical, the student(s) should have a solid conceptual and working knowledge of system dynamics modeling. Furthermore, the student(s) should be very familiar with at least one of the major system dynamics modeling software packages available on the current market. These include Vensim, PowerSim, iThink, as well as others. The student(s) should have a solid understanding of ACE's needs, and more specifically, the Andersen Project Management Fundamentals training game and the Tarek Abdel-Hamid software project management system dynamics model.

The following year, the Ph.D. student(s) will go through a more detailed version of ACE's needs in a system dynamics game engine. This means ACE professionals and the student(s) need to review the Abdel-Hamid model and decide which parts are important, which parts should be deleted, and what other aspects of project management should be added. The level of detail for the PERIOD1 engine should also be discussed. The student(s), under the supervision of Professor Sterman, will use the Vensim system dynamics model ported by the author and make appropriate changes. The implementation of the PERIOD1 engine may be fairly complex, involving many changes to the base model and requiring thorough testing. Professor Sterman will serve as the

expert on this end of the project and make sure that the changes made are both reasonable and effective.

CECI and CAES will work on the other half of the project - creating an effective user interface. Professor Steven Lerman and Professor Richard Larson will serve as the experts and supervisors. There should be at least two research assistants working on this area of the project. During the first year, the students should familiarize themselves with the various multimedia technologies available, especially those involving the internet and distance learning. The students should take at least the introductory system dynamics class at the Sloan School of Management. Although, the students will probably not be deeply involved in the design aspect of the model, the students should be familiar with system dynamics concepts so that they can understand the model they are creating the user interface for. The students do not necessarily need to be computer science majors, but experience with computers will help during major phases of the project such as the implementation of the user interface and porting the game onto an ACE intranet. There is currently a web-based system dynamics modeling software package on the market called PowerSim. PowerSim will be researched at MIT as a potential tool for porting the game engine onto the world wide web.

During the second year, the research assistants will work extensively with ACE professionals and the Sloan Ph.D. candidate(s) on this project. They need to decide which parameters in the game should be hard coded, and which the trainees should be able to modify. The actual user interface will then be created. As soon as parts of the game can be played, volunteers (ACE professionals and MIT students) should be called in to use the game and give feedback.

After the computerized version of the Andersen Project Management Fundamentals training game has been implemented, many more doors should open. Given the gained expertise in system dynamics modeling in game engines, future needs regarding the use of system dynamics for ACE training will be more easily designed and implemented. ACE can use the knowledge of web-based business games and system dynamics to computerize currently manual games as well as to create new game engines. Furthermore, there will also be a vast amount of knowledge gained on effective user interfaces, how trainees learn, and distance learning. CAES and CECI at MIT hope to maintain a close relationship with ACE and collaborate on future projects.

Bibliography

[1] Keys, Benard, and Joseph Wolfe. "The Role of Management Games and Simulations in Education and Research." Journal of Management 1990, Vol. 16, No. 2: 307-336.

[2] Larreche, Jean-Claude. "On Simulations in Business Education and Research." Journal of Business Research.Vol. 15, 1987.

[3] Larson, Richard, et al. "MIT PM Game Proposal." June 17, 1996.

[4] Sterman, John. "Learning in and about Complex Systems." System Dynamics Review. 1994, Vol. 10: 2-3.

[5] Solomon, Jolie. "Now, Simulators for Piloting Companies - Computers Let Managers Test Various Tactics." Wall Street Journal. July 31, 1989: B1.

[6] Hollis, Robert. "Program Lets Users Pilot, Crash Airline Company." MacWEEK. October 16, 1990: 24.

[7] Fiske, Edward B., and Bruce Hammond. "Computer Savvy in the B-Schools." Lotus. September 1989.

[8] Whitestone, Debra. "People Express(A)." Harvard Business School Publishing. May 1995.

[9] Sterman, John. Foresight and National Decisions. University Press of America, 1988

[10] Forrester, Jay. From the Ranch to System Dynamics: An Autobiography. JAI Press,1991.

[11] Arthur, Brian. Increasing Returns and Path Dependence in the Economy. The University of Michigan Press

[12] Kaplan, Robert. "Analog Devices: The Half-Life System." Harvard Business School Publishing, 1990

[13] Forrester, Jay. "Counterintuitive Behavior of Social Systems." Technology Review. January, 1971, Vol. 73, No. 3.

[14] Salpukas, Agis. "For Supertankers, Super Profits." New York Times. December 5th, 1989: D1

[15] Sterman, John. System Dynamics Modeling for Project Management, 1992

[16] Brooks, Frederick Jr.. The Mythical Man-Month. Addison-Wesley Publishing Co.,1995

[17] Smith, Bradley J., and Nghia Nguyen. "Death of a Software Manager: How to Avoid Career Suicide Through Dynamic Software Process Modeling." American Programmer, May 1994

[19] Abdel-Hamid, Tarek, and Stuart E. Madnick. Software Project Dynamics - An Integrated Approach. Prentice-Hall Inc., 1991

Appendix A:

Vensim Model Equations

```
active detection and correction rate = MIN(testing rate*active error density,undetected active errors
/TIME STEP)
    ~ errors/day
    ~     |

time of last exhaustion breakdown =INTEG(Input time of last breakdown-breakdown flush,-1)
    ~ day
    ~     |

exhaust switch = IF THEN ELSE(exhaustion/maximum tolerable exhaustion>=0.1, 1, 0)
    ~ dimensionless
    ~     |

day dimension = 1
    ~ day
    ~     |

tasks discovered = INTEG(rate of discovering tasks-rate of incorporating discovered tasks into project
, 0)
    ~ tasks
    ~     |

rate of discovering tasks =undiscovered job tasks*percent of undiscovered tasks discovered per day/100
    ~ tasks/day
    ~     |

rate of incorporating discovered tasks into project =DELAY3(rate of discovering tasks,average delay in
incorporating discovered tasks
)
    ~ tasks/day
    ~     |

rate of increase in dev man days due to discovered tasks = (rate of incorporating discovered tasks into project
/assumed dev productivity)*fraction of additional tasks added to man days
    ~ (man*day)/day
    ~     |

exhaust inflow = 1
    ~ dimensionless

    |

currently perceived job size in tasks = INTEG(rate of incorporating discovered tasks into project,perceived
job size in dsi
/dsi per task)
    ~ tasks
    ~     |

exhaust variable flush = IF THEN ELSE(breakdown switch=1 :OR: exhaust switch=1, variable that controls
time to de exhaust
/TIME STEP+exhaust inflow,0)
    ~ dimensionless
    ~     |

dev man days = total man days*percent effort assumed needed for dev
    ~ (man*day)
    ~     |
```

Input time of last breakdown =breakdown switch*(Time/TIME STEP)
~ dimensionless
~ |

percent boost in work rate sought = IF THEN ELSE(perceived excess or shortage in man days>=0,handled man days
/(full time equivalent workforce
*(overwork duration threshold+days zidz3)),handled man days/(total man days perceieved still needed
-handled man days+man days zidz3))
~ dimensionless
~ |

testing overhead = (1*dimension factor)/1000
~ (man*day)/dsi
~ |

rate of increase in testing due to discovered tasks = (rate of incorporating discovered tasks into project
/perceived testing productivity)*fraction of additional tasks added to man days
~ (man*day)/day
~ |

total dev time = schedule compression switch*((19*2.5*EXP(0.38*LN(total man days/19*dimensionless man day
)))*schedule compression factor
)+(1*day dimension-schedule compression switch
)*time to dev1
~ day
~ |

initial experienced workforce = team size at beginnng of design
~ man
~ |

scheduled completion date = INTEG(rate of adjusting schedule,total dev time)
~ day
~ |

days zidz2 = 0.001
~ day
~ |

breakdown flush = breakdown switch*(time of last exhaustion breakdown/TIME STEP)
~ dimensionless
~ |

breakdown switch = IF THEN ELSE(overwork duration threshold = 0,1,0)
~ dimensionless
~ |

man day dimension = 1
~ (man*day)
~ |

dimension factor = 1
~ (man*day)/dsi
~ |

days zidz3 = 0.0001

~ day
~ |

total man days =man day switch*(((2.4*EXP(1.05*LN(perceived job size in dsi/1000*dimensionless dsi))
)*19)*(1-man days underestimation fraction
))+(1*man day dimension-man day switch)*total man days1
~ (man*day)
~ |

dimensionless dsi = 1
~ 1/dsi
~ |

undetected passive errors = INTEG(active error retirement rate
+passive error generation rate-passive detection and correction rate
,0)
~ errors
~ |

undiscovered job tasks = INTEG(-rate of discovering tasks,real job size in tasks-currently perceived job size
in tasks
)
~ tasks
~ |

dimensionless man day = 1
~ 1/(man*day)
~ |

variable that controls time to de exhaust = INTEG(exhaust inflow-exhaust variable flush,0)
~ day
~ |

active error density = undetected active errors/(cumulative tasks QAed+tasks zidz)
~ errors/tasks
~ |

active error generation rate = (error escape rate+bad fixes generation rate)/fraction of escaping errors that will
become active
~ errors/day
~ |

active error regeneration rate = software dev rate*SMOOTH(active error density,time to smooth active error
density
*multiplier to active error generation due to error density)
~ errors/day
~ |

active error retirement rate = undetected active errors*active error retiring fraction
~ errors/day
~ |

active error retiring fraction =table active error retirement fraction(percent of job actually worked
)
~ 1/day
~ |

actual fraction of man day on project = INTEG(work rate adjustment rate

,nominal fraction of man days on project)
~ dimensionless
~ |

actual fraction of manpower for QA = IF THEN ELSE(Time = 0,planned fraction of manpower for QA,
planned fraction of manpower for QA*(1+percent adjustement in planned fraction of manpower for QA))
~ dimensionless
~ |

actual rework manpower needed per error = nominal rework manpower needed per error/multiplier to
productivity due to motivation and communication losses
~ (man*day)/errors
~ |

actual testing productivity = cumulative tasks tested/(cumulative testing man days+man days zidz2)
~ tasks/(man*day)
~ |

all errors = cumulative errors reworked in testing phase+cumulative reworked errors during development
+detected errors+potentially detectable errors+undetected active errors+undetected passive errors
~ errors
~ ~:SUPPLEMENTARY
|

all errors reworked in dev and testing = cumulative errors reworked in testing phase+cumulative reworked
errors during development
~ errors
~ ~:SUPPLEMENTARY
|

all errors that escaped and were generated =cumulative errors reworked in testing phase+undetected active
errors
+undetected passive errors
~ errors
~ ~:SUPPLEMENTARY
|

assumed dev productivity = projected dev productivity*weight of projected productivity+perceived dev
productivity
*(1-weight of projected productivity)
~ tasks/(man*day)
~ |

percent of task reported complete = IF THEN ELSE(Time = 0,0,
SMOOTH((100-(man days reported still needed/total job size in man days)*100),
reporting delay))
~ dimensionless
~ ~:SUPPLEMENTARY
|

average daily manpower per staff = 1
~ dimensionless
~ |

average delay in incorporating discovered tasks = 10
~ day
~ |

```
planned fraction of manpower for QA =table planned fraction of manpower for QA(percent of job actually
worked
)*(1+quality objective/100)
    ~ dimensionless
    ~    |

average nominal potential productivity = fraction of experienced workforce*nominal potential productivity
experienced
+(1-fraction of experienced workforce)*nominal potential productivity new
    ~ tasks/(man*day)
    ~    |

average number of errors per task = MAX(potentially detectable errors/(tasks dev+tasks zidz3),0)
    ~ errors/tasks
    ~    |

average QA delay = 10
    ~ day
    ~    |

bad fixes generation rate = percent bad fixes*rework rate
    ~ errors/day
    ~    |

potentially detectable errors = INTEG(+error generation rate
-error detection rate-error escape rate,0)
    ~ errors
    ~    |

projected dev productivity = tasks perceived remaining/(man days perceived remaining for new tasks+man
days zidz
)
    ~ tasks/(man*day)
    ~    |

communication overhead =table communication overhead(total workforce)
    ~ dimensionless
    ~    |

control switch = 1
    ~ dimensionless
    ~ allows us to test policy of no overwork
    |

cumulative dev man days = INTEG(daily manpower for dev and testing*(1-fraction of effort for system
testing
),0)
    ~ (man*day)
    ~    ~:SUPPLEMENTARY
    |

cumulative errors detected = INTEG(error detection rate,0)
    ~ errors
    ~    |

cumulative errors generated directly during working = INTEG(
error generation rate,0)
    ~ errors
```

```
    ~     |

cumulative errors reworked in testing phase = INTEG(active detection and correction rate
+passive detection and correction rate,0)
    ~ errors
    ~     |

cumulative errors that escaped = INTEG(error escape rate,0)
    ~ errors
    ~     ~:SUPPLEMENTARY
    |

cumulative man days expended = INTEG(total daily manpower,0.0001)
    ~ (man*day)
    ~     |

cumulative QA man days = INTEG(daily manpower for QA,0)
    ~ (man*day)
    ~     ~:SUPPLEMENTARY
    |

cumulative rework man days = INTEG(daily manpower allocation for rework,0)
    ~ (man*day)
    ~     ~:SUPPLEMENTARY
    |

cumulative reworked errors during development = INTEG(rework rate,0)
    ~ errors
    ~     |

cumulative tasks dev = INTEG(software dev rate,0)
    ~ tasks
    ~     |

cumulative tasks QAed = INTEG(QA rate-testing rate,0)
    ~ tasks
    ~     |

cumulative tasks tested = INTEG(testing rate,0)
    ~ tasks
    ~     |

cumulative testing man days = INTEG(daily manpower for testing,0)
    ~ (man*day)
    ~     |

cumulative training man days = INTEG(daily manpower for training,0)
    ~ (man*day)
    ~     ~:SUPPLEMENTARY
    |

daily manpower allocation for rework =IF THEN ELSE(Time = 0,0,
MIN(desired error correction rate*perceived rework manpower needed per error
,daily manpower for software production))
    ~ (man*day)/day
    ~     |

daily manpower available after training overhead = total daily manpower-daily manpower for training
```

~ (man*day)/day
~ |

daily manpower for dev and testing =daily manpower for software production-daily manpower allocation for rework
~ (man*day)/day
~ |

daily manpower for QA = MIN((actual fraction of manpower for QA*total daily manpower),0.9*daily manpower available after training overhead
)
~ (man*day)/day
~ |

daily manpower for software dev = daily manpower for dev and testing*(1-fraction of effort for system testing
)
~ (man*day)/day
~ |

daily manpower for software production =daily manpower available after training overhead-daily manpower for QA
~ (man*day)/day
~ |

daily manpower for testing = daily manpower for dev and testing*fraction of effort for system testing
~ (man*day)/day
~ |

software dev productivity = potential productivity*multiplier to productivity due to motivation and communication losses
~ tasks/(man*day)
~ |

day zidz2 = 0.001
~ day
~ |

delay in adjusting job size in man days = table delay in adjusting job size in man days(time remaining
)
~ day
~ |

desired error correction rate =IF THEN ELSE(Time = 0,0,detected errors/desired rework delay)
~ errors/day
~ |

desired rework delay = 15
~ day
~ |

detected errors = INTEG(error detection rate-rework rate,0)
~ errors
~ |

dsi per KDSI = 1000
~ dsi/KDSI
~ |

dsi per task = 60
~ dsi/tasks
~ |

effect of exhaustion on overwork duration threshold =table effect of exhaustion on overwork duration threshold
(exhaustion/maximum tolerable exhaustion)
~ dimensionless
~ |

effect of work rate sought = IF THEN ELSE(work rate sought>=actual fraction of man day on project,1, 0.75)
~ dimensionless
~ |

error density =(average number of errors per task*1000/dsi per task)
~ errors/dsi
~ |

error detection rate = MIN(potential error detection rate,potentially detectable errors/TIME STEP)
~ errors/day
~ |

error escape rate = QA rate*average number of errors per task
~ errors/day
~ |

error generation rate = software dev rate*errors per task
~ errors/day
~ |

errors per task = multiplier to error generation due to schedule pressure*multiplier to error generation due to workforce mix
*nominal errors committed per task
~ errors/tasks
~ |

errors zidz2 = 0.001
~ errors
~ |

exhaustion = INTEG(+rate of increase in exhaustion level-rate of depletion in exhaustion level ,0)
~ exhaust units
~ |

exhaustion depletion delay time = 20
~ day
~ |

experienced transfer rate = MIN(experienced workforce/TIME STEP, transfer rate-newly hired transfer rate)
~ man/day
~ |

experienced workforce = INTEG(-experienced transfer rate+workforce assimilation rate -quit rate,initial experienced workforce)

~ man
~ |

fraction of additional tasks added to man days = table fraction of additional tasks added to man days
(relative size of discovered tasks/(maximum relative size of additions tolerated without adding to project man
days
+0.001))
~ dimensionless
~ |

fraction of effort for system testing = table fraction of effort for system testing(tasks perceived remaining
/currently perceived job size in tasks)
~ dimensionless
~ |

fraction of escaping errors that will become active = table fraction of escaping errors that will become active
(percent of job actually worked)
~ dimensionless
~ |

fraction of experienced workforce =experienced workforce/total workforce
~ dimensionless
~ |

full time equivalent workforce = total workforce*average daily manpower per staff
~ man
~ |

full time experienced workforce = experienced workforce*average daily manpower per staff
~ man
~ |

handled man days = IF THEN ELSE(perceived excess or shortage in man days>=0,MIN(maximum man days
shortage handled
,perceived excess or shortage in man days),-man day excesses that will be absorbed)*control switch
~ (man*day)
~ |

table willingness to change workforce 2 ([(0.86,0)-(1,1)],(0.86,0)
,(0.88,0.1),(0.9,0.2),(0.92,0.35),(0.94,0.6),(0.96,0.7),(0.98,0.77)
,(1,0.8))
~ dimensionless
~ |

tasks dev = INTEG(software dev rate- QA rate,0)
~ tasks
~ |

indicated completion date = Time+time perceived still required
~ day
~ |

indicated workforce level = (man days remaining/(time remaining+day zidz2))/average daily manpower per
staff
~ man
~ |

initial understaffing factor = 0.5

~ dimensionless
~ |

man day excesses that will be absorbed = MAX(0,table man day excesses that will be absorbed(total man days perceieved still needed
/man days remaining)*man days remaining-total man days perceieved still needed)
~ (man*day)
~ |

man day switch = 1
~ man*day
~ |

man days perceived needed to rework detected errors = detected errors*perceived rework manpower needed per error
~ (man*day)
~ |

man days perceived remaining for new tasks = MAX(0,man days remaining-man days perceived needed to rework detected errors
-man days perceived still needed for testing)
~ (man*day)
~ |

man days perceived still needed for testing =tasks remaining to be tested/perceived testing productivity
~ (man*day)
~ |

man days remaining =MAX(0.0001,total job size in man days-cumulative man days expended)
~ (man*day)
~ |

man days reported still needed = man days remaining+reported shortage excess in man days
~ (man*day)
~ |

man days underestimation fraction = 0
~ dimensionless
~ |

man days zidz = 0.1
~ (man*day)
~ |

man days zidz2 = 0.001
~ (man*day)
~ |

man days zidz3 = 0.0001
~ (man*day)
~ |

maximum boost in man hours = 1
~ dimensionless
~ |

maximum man days shortage handled = (overwork duration threshold*full time equivalent workforce*maximum boost in man hours

)*willingness to overwork
~ (man*day)
~ |

maximum relative size of additions tolerated without adding to project man days = 0.01
~ dimensionless
~ |

maximum schedule completion date extension = 1e+006
~ dimensionless
~ |

maximum tolerable completion date =maximum schedule completion date extension*total dev time
~ day
~ |

maximum tolerable exhaustion = 50
~ exhaust units
~ |

multiplier to active error generation due to error density = table multiplier to active error generation due to error density
(SMOOTH(active error density*1000/dsi per task,time to smooth active error density))
~ dimensionless
~ |

multiplier to detection effort due to error density = table multiplier to detection effort due to error density
(error density)
~ dimensionless
~ |

multiplier to error generation due to schedule pressure = table multiplier to error generation due to schedule pressure
(schedule pressure
)
~ dimensionless
~ |

multiplier to error generation due to workforce mix = table muliplier to error generation due to workforce mix
(fraction of experienced workforce
)
~ dimensionless
~ |

multiplier to potential productivity due to learning = table multiplier to potential productivity due to learning
(percent of job actually worked)
~ dimensionless
~ |

multiplier to productivity due to motivation and communication losses = actual fraction of man day on project
*(1-communication overhead)
~ dimensionless
~ |

multiplier to productivity weight due to dev = table multiplier to productivity weight due to dev(

```
percent of perceived job dev/100)
    ~ dimensionless
    ~       |

multiplier to productivity weight due to resource expenditure =table multiplier to productivity weight due to
resource expenditure
(1- man days perceived remaining for new tasks/(total job size in man days-planned testing size in man days
))
    ~ dimensionless
    ~       |

newly hired transfer rate =MIN(transfer rate,newly hired work force/TIME STEP)
    ~ man/day
    ~       |

undetected active errors = INTEG(+active error generation rate
+active error regeneration rate-active detection and correction rate
-active error retirement rate,0)
    ~ errors
    ~       |

nominal errors committed per dsi = table nominal errors committed per KDSI(percent of job actually worked
)*(1/dsi per KDSI)
    ~ errors/dsi
    ~       |

nominal errors committed per task =nominal errors committed per dsi*dsi per task/1000
    ~ errors/tasks
    ~       |

nominal fraction of man days on project = 0.6
    ~ dimensionless
    ~       |

nominal overwork duration threshold = table nominal overwork duration threshold(time remaining)
    ~ day
    ~       |

nominal potential productivity experienced = 1
    ~ tasks/(man*day)
    ~       |

nominal potential productivity new = 0.5
    ~ tasks/(man*day)
    ~       |

nominal QA manpower needed per error = table nominal QA manpower needed per error(percent of job
actually worked
)
    ~ (man*day)/errors
    ~       |

nominal rework manpower needed per error = table nominal rework manpower needed per error(percent of
job actually worked
)
    ~ (man*day)/errors
    ~       |
```

normal work rate adjustment delay = table normal work rate adjustment delay(time remaining)
~ day
~ |

overwork duration threshold =nominal overwork duration threshold*effect of exhaustion on overwork duration threshold
~ day
~ |

passive detection and correction rate = MIN(testing rate*passive error density,undetected passive errors /TIME STEP)
~ errors/day
~ |

passive error density = undetected passive errors/(cumulative tasks QAed+tasks zidz3)
~ errors/tasks
~ |

passive error generation rate = (error escape rate+bad fixes generation rate)*(1-fraction of escaping errors that will become active
)
~ errors/day
~ |

perceived dev productivity = cumulative tasks dev/(cumulative man days expended-cumulative testing man days
)
~ tasks/(man*day)
~ |

perceived excess or shortage in man days =total man days perceieved still needed-man days remaining
~ (man*day)
~ |

perceived job size in dsi = real job size in dsi*(1-tasks underestimation fraction)
~ dsi
~ |

perceived rework manpower needed per error = INTEG((actual rework manpower needed per error-perceived rework manpower needed per error
)/time to adjust perceived rework manpower needed per error,0.5)
~ (man*day)/errors
~ |

perceived size of discovered tasks in man days = tasks discovered/assumed dev productivity
~ (man*day)
~ |

perceived testing productivity = SMOOTH(IF THEN ELSE(0>=cumulative tasks tested,planned testing productivity
,actual testing productivity),time to smooth testing productivity
)
~ tasks/(man*day)
~ |

percent adjustement in planned fraction of manpower for QA =table percent adjustment in planned fraction of manpower for QA
(schedule pressure)

~ dimensionless
~ |

percent bad fixes = 0.075
~ dimensionless
~ |

percent effort assumed needed for dev = 0.8
~ dimensionless
~ |

percent errors detected = 100*cumulative errors detected/(cumulative errors generated directly during working +errors zidz2)
~ dimensionless
~ ~:SUPPLEMENTARY
|

percent of dev perceived complete = SMOOTHI(MAX((100-((man days reported still needed-man days perceived still needed for testing
)/(total job size in man days-planned testing size in man days))*100),percent of dev perceived complete),reporting delay, 0)
~ dimensionless
~ |

percent of job actually worked =cumulative tasks dev/real job size in tasks
~ dimensionless
~ |

percent of perceived job dev = (cumulative tasks dev/currently perceived job size in tasks)*100
~ dimensionless
~ |

table willingness to change workforce 1 ([(0,0)-(3,1.5)],(0,0)
,(0.3,0),(0.6,0.1),(0.9,0.4),(1.2,0.85),(1.5,1),(1.8,1),(2.1,1)
,(2.4,1),(2.7,1),(3,1))
~ dimensionless
~ |

percent of tasks tested = cumulative tasks tested/currently perceived job size in tasks
~ dimensionless
~ ~:SUPPLEMENTARY
|

percent of undiscovered tasks discovered per day = table percent of undiscovered tasks discovered per day (percent of perceived job dev)
~ 1/day
~ |

planned testing productivity = currently perceived job size in tasks/planned testing size in man days
~ tasks/(man*day)
~ |

planned testing size in man days = INTEG(rate of increase in testing due to discovered tasks+(rate of adjusting job size in man days
/TIME STEP)
*IF THEN ELSE(fraction of effort for system testing>=0.9,1,0),testing man days)
~ (man*day)
~ |

potential error detection rate =daily manpower for QA/QA manpower needed to detect an error
~ errors/day
~ |

potential productivity = average nominal potential productivity*multiplier to potential productivity due to learning
~ tasks/(man*day)
~ |

tasks zidz3 = 0.0001
~ tasks
~ |

team size = (total man days/total dev time)/average daily manpower per staff
~ man
~ |

QA manpower needed to detect an error = nominal QA manpower needed per error*(1/multiplier to productivity due to motivation and communication losses
)*multiplier to detection effort due to error density
~ (man*day)/errors
~ |

QA rate =DELAY3(software dev rate,average QA delay)
~ tasks/day
~ |

quality objective = 0
~ dimensionless
~ |

testing manpower needed per task = (testing overhead*dsi per task/1000+testing manpower needed per error *(passive error density+active error density))/multiplier to productivity due to motivation and communication losses
~ (man*day)/tasks
~ |

rate of adjusting job size in man days = (man days reported still needed+cumulative man days expended -total job size in man days)/delay in adjusting job size in man days
~ (man*day)/day
~ |

rate of adjusting schedule = (indicated completion date-scheduled completion date)/schedule adjustment time
~ dimensionless
~ |

rate of depletion in exhaustion level = IF THEN ELSE(0>=rate of increase in exhaustion level,exhaustion /exhaustion depletion delay time,0)
~ exhaust units/day
~ |

rate of increase in exhaustion level = table rate of increase in exhaustion level((1-actual fraction of man day on project
)/(1-nominal fraction of man days on project))
~ exhaust units/day
~ |

```
real job size in dsi = 64000
    ~ dsi
    ~     |

real job size in tasks =real job size in dsi/dsi per task
    ~ tasks
    ~     |

relative size of discovered tasks = perceived size of discovered tasks in man days/(man days perceived
remaining for new tasks
+ man days zidz3)
    ~ dimensionless
    ~     |

reported shortage excess in man days = perceived excess or shortage in man days-handled man days
    ~ (man*day)
    ~     |

reporting delay = 10
    ~ day
    ~     |

rework rate =daily manpower allocation for rework/actual rework manpower needed per error
    ~ errors/day
    ~     |

schedule adjustment time =table schedule adjustment time(time remaining)
    ~ day
    ~     |

schedule compression factor = 1
    ~ dimensionless
    ~     |

schedule compression switch = 1
    ~ day
    ~     |

schedule pressure = (total man days perceieved still needed-man days remaining)/man days remaining
    ~ dimensionless
    ~     |

table multiplier to productivity weight due to dev ([(0,0)-(1,1.5)],
(0,1),(0.1,1),(0.2,1),(0.3,1),(0.4,1),(0.5,1),(0.6,0.975),(0.7,0.9)
,(0.8,0.75),(0.9,0.5),(1,0) )
    ~ dimensionless
    ~     |

software dev rate = IF THEN ELSE(Time = 0,0,
MIN(daily manpower for software dev*software dev productivity,
tasks perceived remaining/TIME STEP))
    ~ tasks/day
    ~     |

table active error retirement fraction ([(0,0)-(1,1)],(0,0),
(0.1,0),(0.2,0),(0.3,0),(0.4,0.01),(0.5,0.02),(0.6,0.03),(0.7,0.04)
,(0.8,0.1),(0.9,0.3),(1,1) )
    ~ 1/day
```

```
    ~     |

table communication overhead (
[(0,0)-(35,1)],(0,0),(5,0.015),(10,0.06),(15,0.135),(20,0.24)
,(25,0.375),(30,0.54) )
    ~ dimensionless
    ~     |

table delay in adjusting job size in man days ([(0,0)-(22,5)],
(0,0.5),(20,3) )
    ~ day
    ~     |

table effect of exhaustion on overwork duration threshold ([(0,0)-(1,1)],
(0,1),(0.1,0.9),(0.2,0.8),(0.3,0.7),(0.4,0.6),(0.5,0.5),(0.6,0.4)
,(0.7,0.3),(0.8,0.2),(0.9,0.1),(1,0) )
    ~ dimensionless
    ~     |

table fraction of additional tasks added to man days ([(0,0)-(2,1)],
(0,0),(0.2,0),(0.4,0),(0.6,0),(0.8,0),(1,0),(1.2,0.7),(1.4,0.9)
,(1.6,0.975),(1.8,1),(2,1) )
    ~ dimensionless
    ~     |

table fraction of effort for system testing  ([(0,0)-(0.25,1)],
(0,1),(0.04,0.5),(0.08,0.28),(0.12,0.15),(0.16,0.05),(0.2,0)
)
    ~ dimensionless
    ~     |

table fraction of escaping errors that will become active ([(0,0)-(1,1.2)],
(0,1),(0.1,1),(0.2,1),(0.3,1),(0.4,0.95),(0.5,0.85),(0.6,0.5)
,(0.7,0.2),(0.8,0.075),(0.9,0),(1,0) )
    ~ dimensionless
    ~     |

table man day excesses that will be absorbed ([(0,0)-(1,1)],(0,0)
,(0.1,0.2),(0.2,0.4),(0.3,0.55),(0.4,0.7),(0.5,0.8),(0.6,0.9)
,(0.7,0.95),(0.8,1),(0.9,1),(1,1) )
    ~ dimensionless
    ~     |

table muliplier to error generation due to workforce mix ([(0,0)-(1,2.5)],
(0,2),(0.2,1.8),(0.4,1.6),(0.6,1.4),(0.8,1.2),(1,1) )
    ~ dimensionless
    ~     |

table multiplier to active error generation due to error density (
[(0,0)-(100,10)],(0,1),(10,1.1),(20,1.2),(30,1.325),(40,1.45)
,(50,1.6),(60,2),(70,2.5),(80,3.25),(90,4.35),(100,6) )
    ~ dimensionless
    ~     |

table multiplier to detection effort due to error density ([(0,0)-(10,60)],
(0,50),(1,36),(2,26),(3,17.5),(4,10),(5,4),(6,1.75),(7,1.2),
(8,1),(9,1),(10,1) )
    ~ dimensionless
```

```
    ~     |

table multiplier to error generation due to schedule pressure (
[(-0.6,0)-(1.2,2)],(-0.4,0.9),(-0.2,0.94),(0,1),(0.2,1.05),(0.4,1.14)
,(0.6,1.24),(0.8,1.36),(1,1.5) )
    ~ dimensionless
    ~     |

table multiplier to potential productivity due to learning (
[(0,1)-(1,1.5)],(0,1),(0.1,1.0125),(0.2,1.0325),(0.3,1.055),
(0.4,1.091),(0.5,1.15),(0.6,1.2),(0.7,1.22),(0.8,1.245),(0.9,1.25)
,(1,1.25) )
    ~ dimensionless
    ~     |

transfer rate = MAX(0,-workforce gap/transfer delay)
    ~ man/day
    ~     |

table multiplier to productivity weight due to resource expenditure (
[(0,0)-(1,1.5)],(0,1),(0.1,1),(0.2,1),(0.3,1),(0.4,1),(0.5,1)
,(0.6,0.975),(0.7,0.9),(0.8,0.75),(0.9,0.5),(1,0) )
    ~ dimensionless
    ~     |

table nominal errors committed per KDSI  ([(0,0)-(1,30)],(0,25)
,(0.2,23.86),(0.4,21.59),(0.6,15.9),(0.8,13.6),(1,12.5) )
    ~ errors/KDSI
    ~     |

table nominal overwork duration threshold ([(0,0)-(60,60)],(0,0)
,(10,10),(20,20),(30,30),(40,40),(50,50) )
    ~ day
    ~     |

table nominal QA manpower needed per error ([(0,0)-(1,0.5)],
(0,0.4),(0.1,0.4),(0.2,0.39),(0.3,0.375),(0.4,0.35),(0.5,0.3)
,(0.6,0.25),(0.7,0.225),(0.8,0.21),(0.9,0.2),(1,0.2) )
    ~ (man*day)/errors
    ~     |

table nominal rework manpower needed per error (
[(0,0)-(1,1)],(0,0.6),(0.2,0.575),(0.4,0.5),(0.6,0.4),(0.8,0.325)
,(1,0.3) )
    ~ (man*day)/errors
    ~     |

table normal work rate adjustment delay  ([(0,0)-(40,12)],(0,2)
,(5,3.5),(10,5),(15,6.5),(20,8),(25,9.5),(30,10) )
    ~ day
    ~     |

table percent adjustment in planned fraction of manpower for QA (
[(0,-0.6)-(0.5,1)],(0,0),(0.1,-0.025),(0.2,-0.15),(0.3,-0.35)
,(0.4,-0.475),(0.5,-0.5) )
    ~ dimensionless
    ~     |
```

```
table percent of undiscovered tasks discovered per day ([(0,-1)-(100,100)],
(0,0),(20,0.4),(40,2.5),(60,5),(80,10),(100,100) )
    ~ 1/day
    ~       |

table planned fraction of manpower for QA    ([(0,0)-(1,1)],
(0,0.15),(0.1,0.15),(0.2,0.15),(0.3,0.15),(0.4,0.15),(0.5,0.15)
,(0.6,0.15),(0.7,0.15),(0.8,0.15),(0.9,0.15),(1,0) )
    ~ dimensionless
    ~       |

table rate of increase in exhaustion level  ([(-1,0)-(1,3)],
(-0.5,2.5),(-0.4,2.2),(-0.3,1.9),(-0.2,1.6),(-0.1,1.3),(0,1.15)
,(0.1,0.9),(0.2,0.8),(0.3,0.7),(0.4,0.6),(0.5,0.5),(0.6,0.4)
,(0.7,0.3),(0.8,0.2),(0.9,0),(1,0) )
    ~ exhaust units/day
    ~       |

table schedule adjustment time  ([(0,0)-(5,5)],(0,0.5),(5,5)
 )
    ~ day
    ~       |

testing rate = MIN(cumulative tasks QAed/TIME STEP,daily manpower for testing/testing manpower
needed per task
)
    ~ tasks/day
    ~       |

time perceived still required = man days remaining/(workforce level sought*average daily manpower per staff
)
    ~ day
    ~       |

time remaining =MAX(scheduled completion date-Time, 0)
    ~ day
    ~       |

tasks perceived remaining = currently perceived job size in tasks-cumulative tasks dev
    ~ tasks
    ~       |

tasks remaining to be tested = currently perceived job size in tasks-cumulative tasks tested
    ~ tasks
    ~       |

tasks underestimation fraction = 0
    ~ dimensionless
    ~       |

tasks zidz = 0.1
    ~ tasks
    ~       |

willingness to change workforce = MAX(willingness to change workforce 1,willingness to change
workforce 2
)
    ~ dimensionless
```

~ |

willingness to change workforce 1 = table willingness to change workforce 1(time remaining/(hiring delay +average assimilation delay))
~ dimensionless
~ |

team size at beginnng of design = initial understaffing factor*team size
~ man
~ ~:SUPPLEMENTARY
|

testing man days = (1-percent effort assumed needed for dev)*total man days
~ (man*day)
~ |

testing manpower needed per error = 0.15
~ (man*day)/errors
~ |

work rate adjustment rate =(work rate sought-actual fraction of man day on project)/work rate adjustment delay
~ 1/day
~ |

work rate sought = (1+percent boost in work rate sought)*nominal fraction of man days on project
~ dimensionless
~ |

total daily manpower = total workforce*average daily manpower per staff
~ (man*day)/day
~ |

workforce gap = workforce level sought-total workforce
~ man
~ |

workforce level needed = MIN(willingness to change workforce*indicated workforce level+total workforce *(1-willingness to change workforce),indicated workforce level)
~ man
~ |

time to adjust perceived rework manpower needed per error = 10
~ day
~ |

time to dev1 = 0
~ dimensionless
~ |

time to smooth active error density = 40
~ day
~ |

time to smooth testing productivity = 50
~ day
~ |

weight of projected productivity =multiplier to productivity weight due to dev*multiplier to productivity weight due to resource expenditure
~ dimensionless
~ |

total job size in man days = INTEG(rate of adjusting job size in man days
+rate of increase in dev man days due to discovered tasks+rate of increase in testing due to discovered tasks ,dev man days+testing man days)
~ (man*day)
~ |

total man days perceieved still needed =man days perceived still needed for testing+man days perceived needed to rework detected errors
+total man days perceived still needed for new tasks
~ (man*day)
~ |

total man days perceived still needed for new tasks = tasks perceived remaining/assumed dev productivity
~ (man*day)
~ |

total man days1 = 0
~ dimensionless
~ |

workforce level sought =MIN(ceiling on total workforce,workforce level needed)
~ man
~ |

willingness to overwork = IF THEN ELSE(Time>=time of last exhaustion breakdown+variable that controls time to de exhaust
, 1,0)
~ dimensionless
~ |

work rate adjustment delay = effect of work rate sought*normal work rate adjustment delay
~ day
~ |

willingness to change workforce 2 = table willingness to change workforce 2(scheduled completion date /maximum tolerable completion date)
~ dimensionless
~ |

average assimilation delay = 80
~ day
~ |

average employment time = 673
~ day
~ |

ceiling on new hires = full time experienced workforce*most new hires per full time experienced staff
~ man
~ |

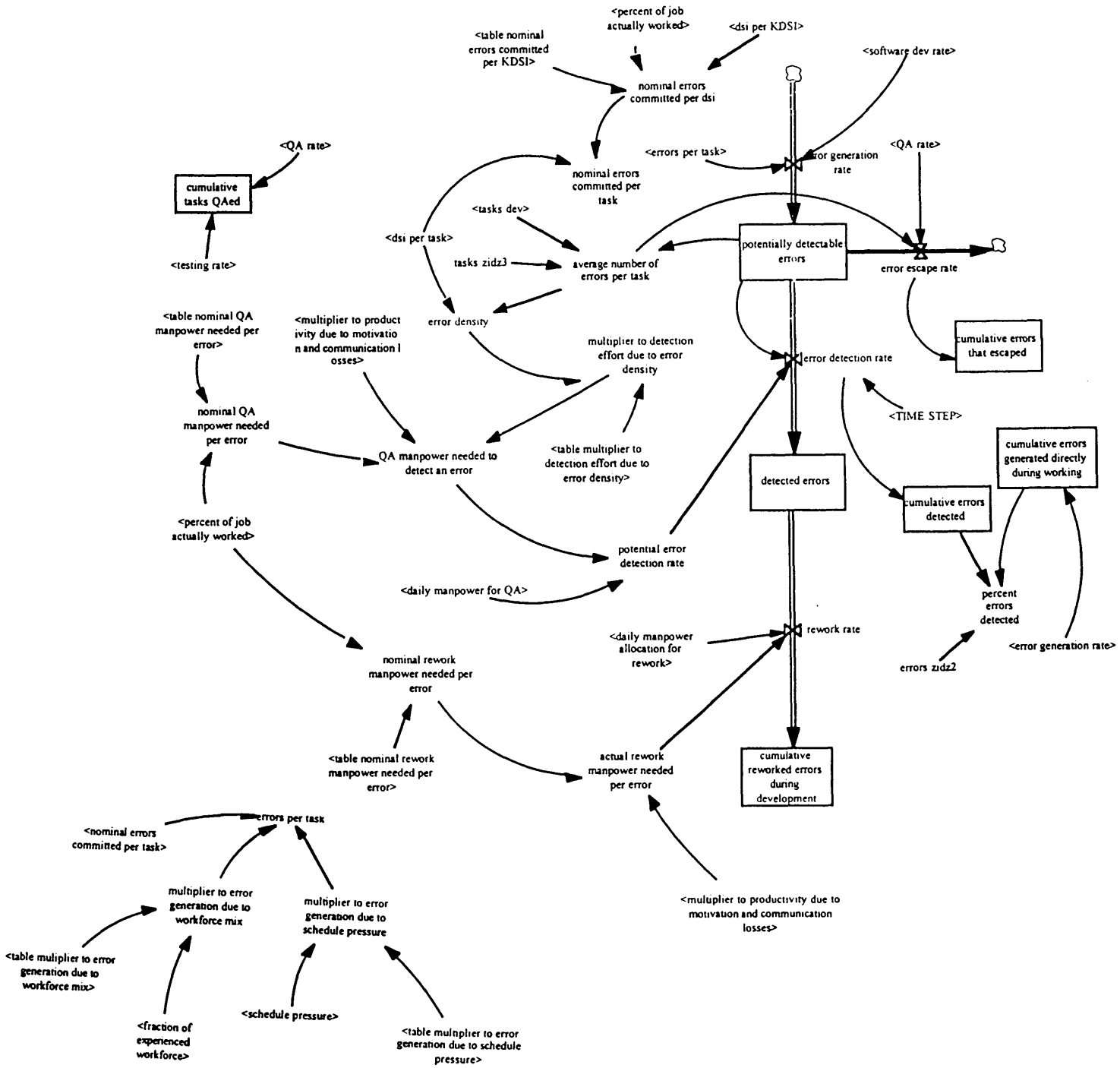
ceiling on total workforce = ceiling on new hires+experienced workforce
~ man

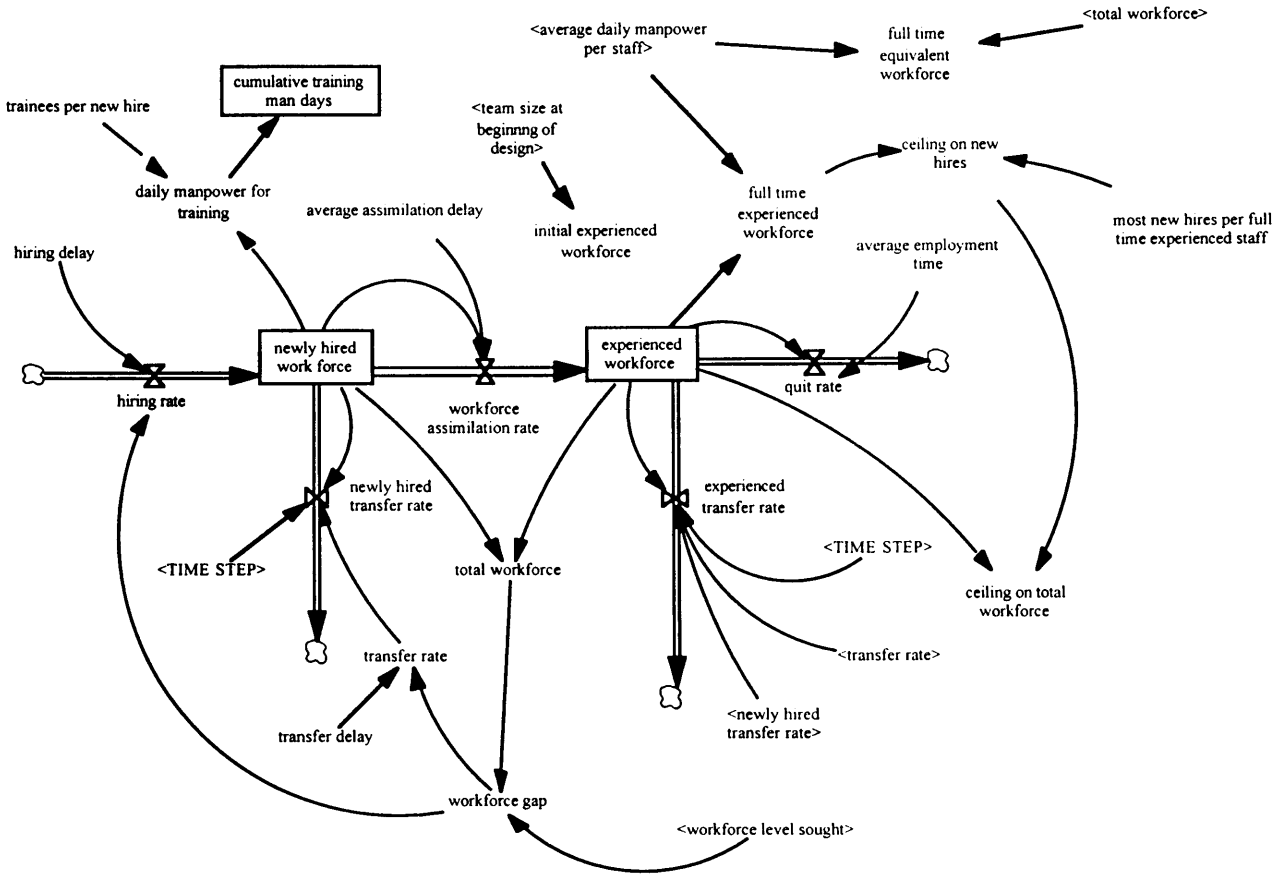
```
    ~      |

daily manpower for training = newly hired work force*trainees per new hire
    ~ (man*day)/day
    ~      |

hiring delay = 40
    ~ day
    ~      |

hiring rate = MAX(0, workforce gap/hiring delay)
    ~ man/day
    ~      |

most new hires per full time experienced staff = 3
    ~ man/man
    ~      |

newly hired work force = INTEG( hiring rate-newly hired transfer rate-workforce assimilation rate,0)
    ~ man
    ~      |

quit rate =experienced workforce/average employment time
    ~ man/day
    ~      |

total workforce = experienced workforce+newly hired work force
    ~ man
    ~      |

trainees per new hire = 0.2
    ~ dimensionless
    ~      |

transfer delay = 10
    ~ day
    ~      |

workforce assimilation rate = newly hired work force/average assimilation delay
    ~ man/day
    ~      |

**********************************************************
    .Control
**********************************************************~
Simulation Control Paramaters
     |

FINAL TIME  = 100
    ~ day
    ~ The final time for the simulation.
    |

INITIAL TIME  = 0
    ~ day
    ~ The initial time for the simulation.
    |
```

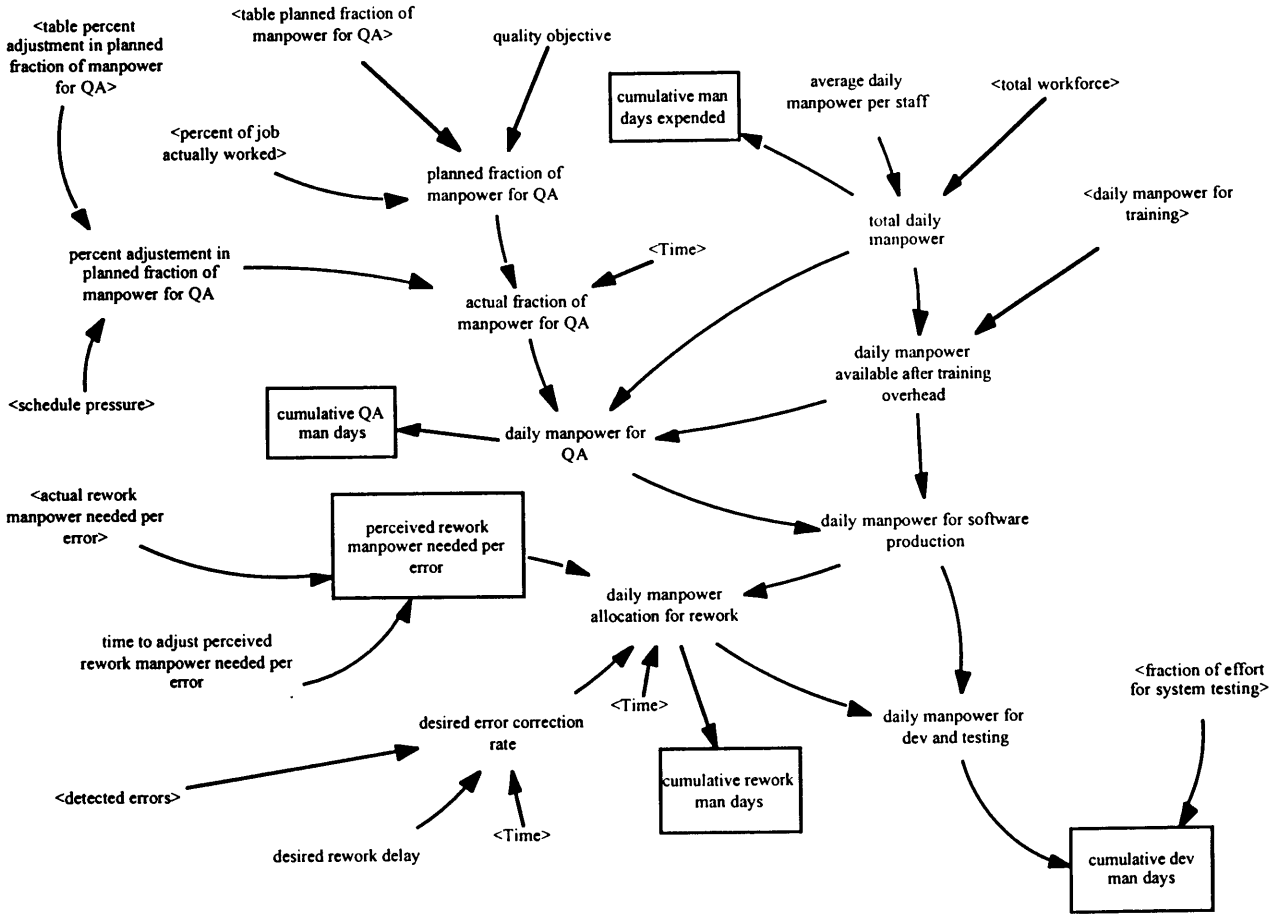
```
SAVEPER =
      TIME STEP
    ~ day
    ~ The frequency with which output is stored.
    |

TIME STEP = 0.5
    ~ day
    ~ The time step for the simulation.
    |
```

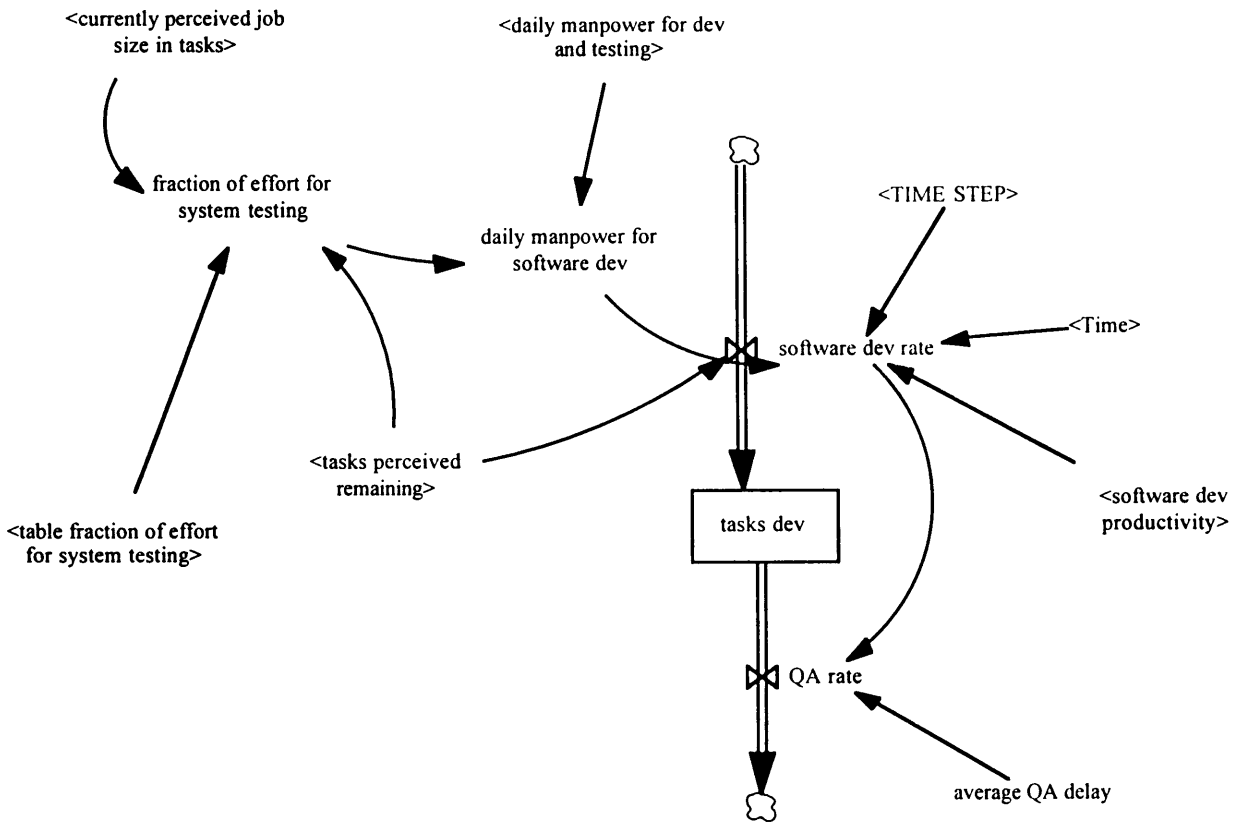
Appendix B:

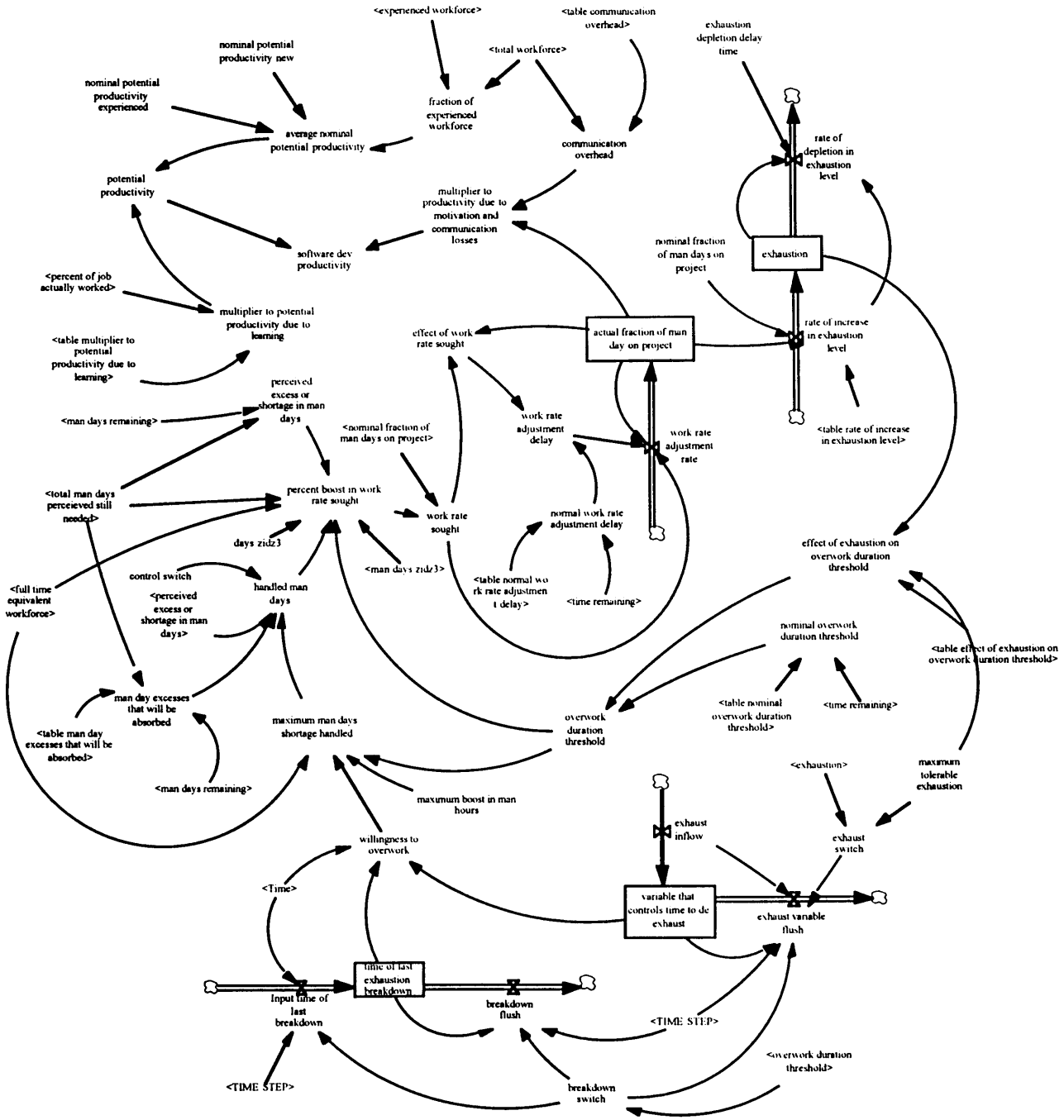
Vensim Model

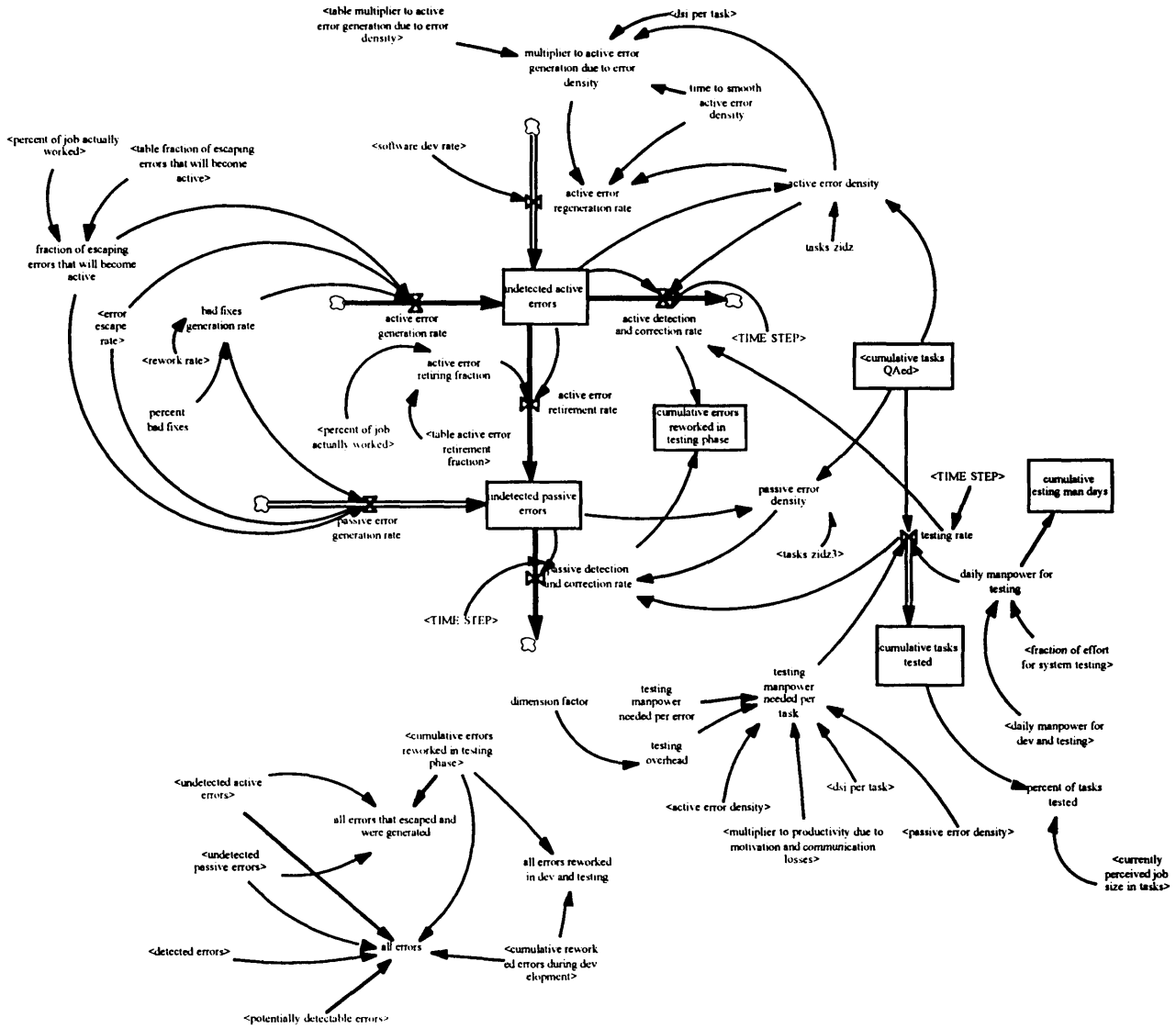

<percent of job actually worked>
<dsi per KDSI>
<table nominal errors committed per KDSI>
<software dev rate>
nominal errors committed per dsi
<QA rate>
<errors per task>
<QA rate>
cumulative tasks QAed
nominal errors committed per task
<tasks dev>
<dsi per task>
potentially detectable errors
<testing rate>
tasks zidz3
average number of errors per task
error escape rate
<table nominal QA manpower needed per error>
<multiplier to productivity due to motivation and communication losses>
error density
multiplier to detection effort due to error density
cumulative errors that escaped
error detection rate
<TIME STEP>
nominal QA manpower needed per error
QA manpower needed to detect an error
<table multiplier to detection effort due to error density>
detected errors
cumulative errors generated directly during working
cumulative errors detected
<percent of job actually worked>
potential error detection rate
percent errors detected
<daily manpower for QA>
<daily manpower allocation for rework>
rework rate
<error generation rate>
errors zidz2
nominal rework manpower needed per error
<table nominal rework manpower needed per error>
actual rework manpower needed per error
cumulative reworked errors during development
errors per task
<nominal errors committed per task>
multiplier to error generation due to workforce mix
multiplier to error generation due to schedule pressure
<multiplier to productivity due to motivation and communication losses>
<table multiplier to error generation due to workforce mix>
<fraction of experienced workforce>
<schedule pressure>
<table multiplier to error generation due to schedule pressure>

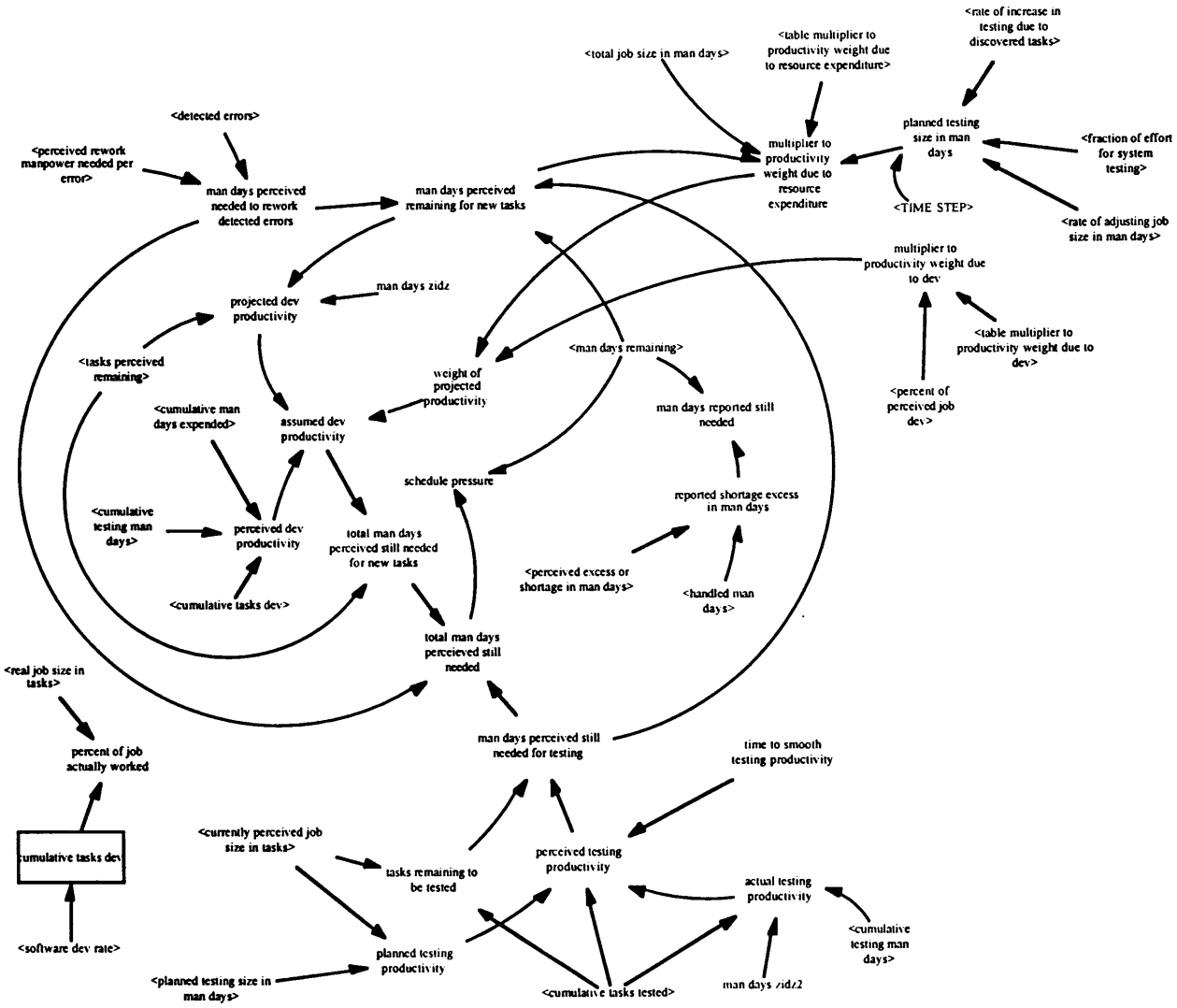

<average daily manpower per staff>
full time equivalent workforce
<total workforce>
cumulative training man days
trainees per new hire
<team size at beginnng of design>
ceiling on new hires
daily manpower for training
average assimilation delay
full time experienced workforce
most new hires per full time experienced staff
initial experienced workforce
hiring delay
average employment time
newly hired work force
experienced workforce
hiring rate
quit rate
workforce assimilation rate
newly hired transfer rate
experienced transfer rate
<TIME STEP>
total workforce
<TIME STEP>
ceiling on total workforce
transfer rate
<transfer rate>
<newly hired transfer rate>
transfer delay
workforce gap
<workforce level sought>

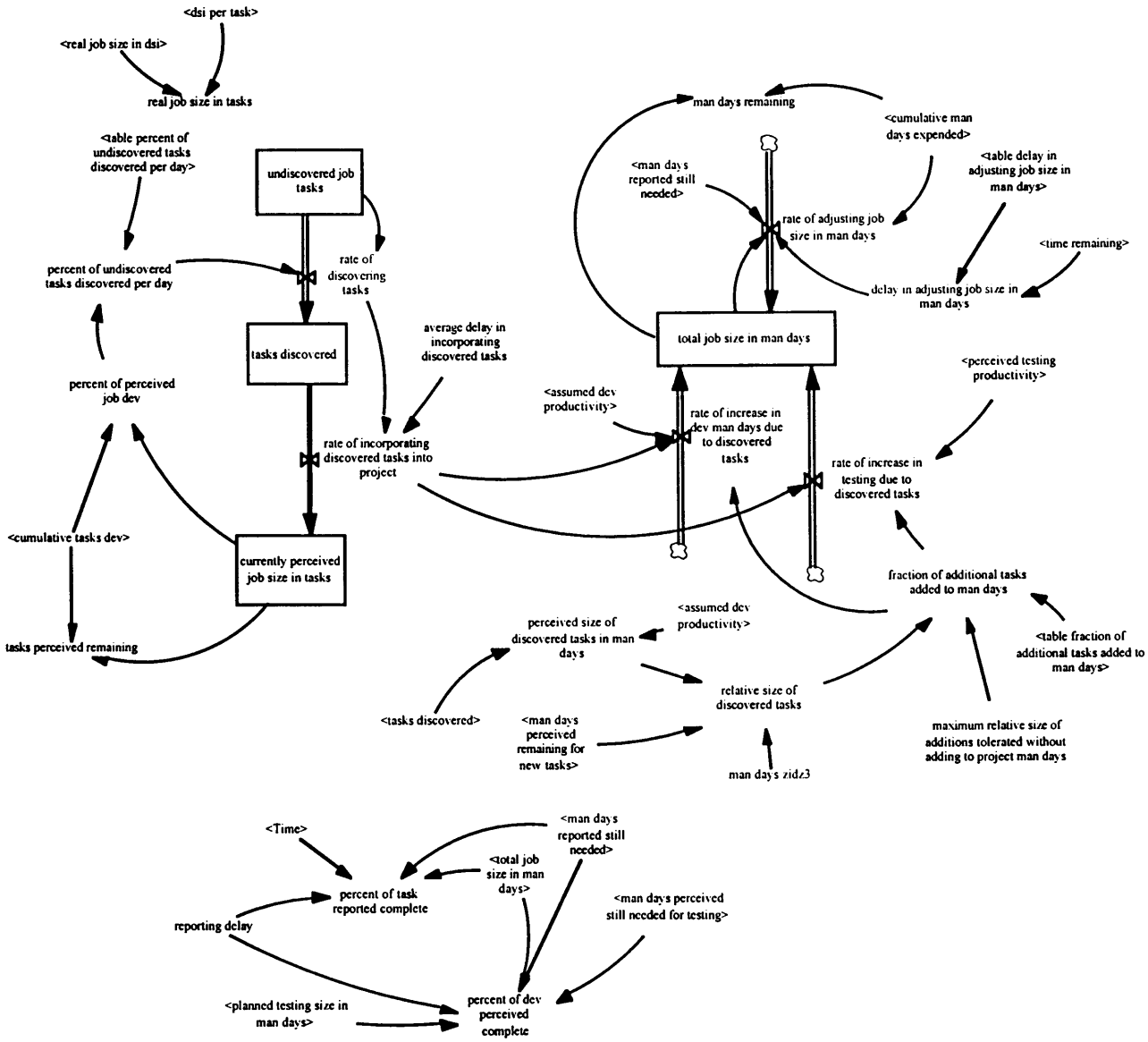

<table percent adjustment in planned fraction of manpower for QA>
<table planned fraction of manpower for QA>
quality objective
cumulative man days expended
average daily manpower per staff
<total workforce>
<percent of job actually worked>
planned fraction of manpower for QA
total daily manpower
<daily manpower for training>
percent adjustement in planned fraction of manpower for QA
<Time>
actual fraction of manpower for QA
daily manpower available after training overhead
<schedule pressure>
cumulative QA man days
daily manpower for QA
<actual rework manpower needed per error>
perceived rework manpower needed per error
daily manpower for software production
daily manpower allocation for rework
time to adjust perceived rework manpower needed per error
<Time>
<fraction of effort for system testing>
desired error correction rate
daily manpower for dev and testing
<detected errors>
cumulative rework man days
<Time>
desired rework delay
cumulative dev man days

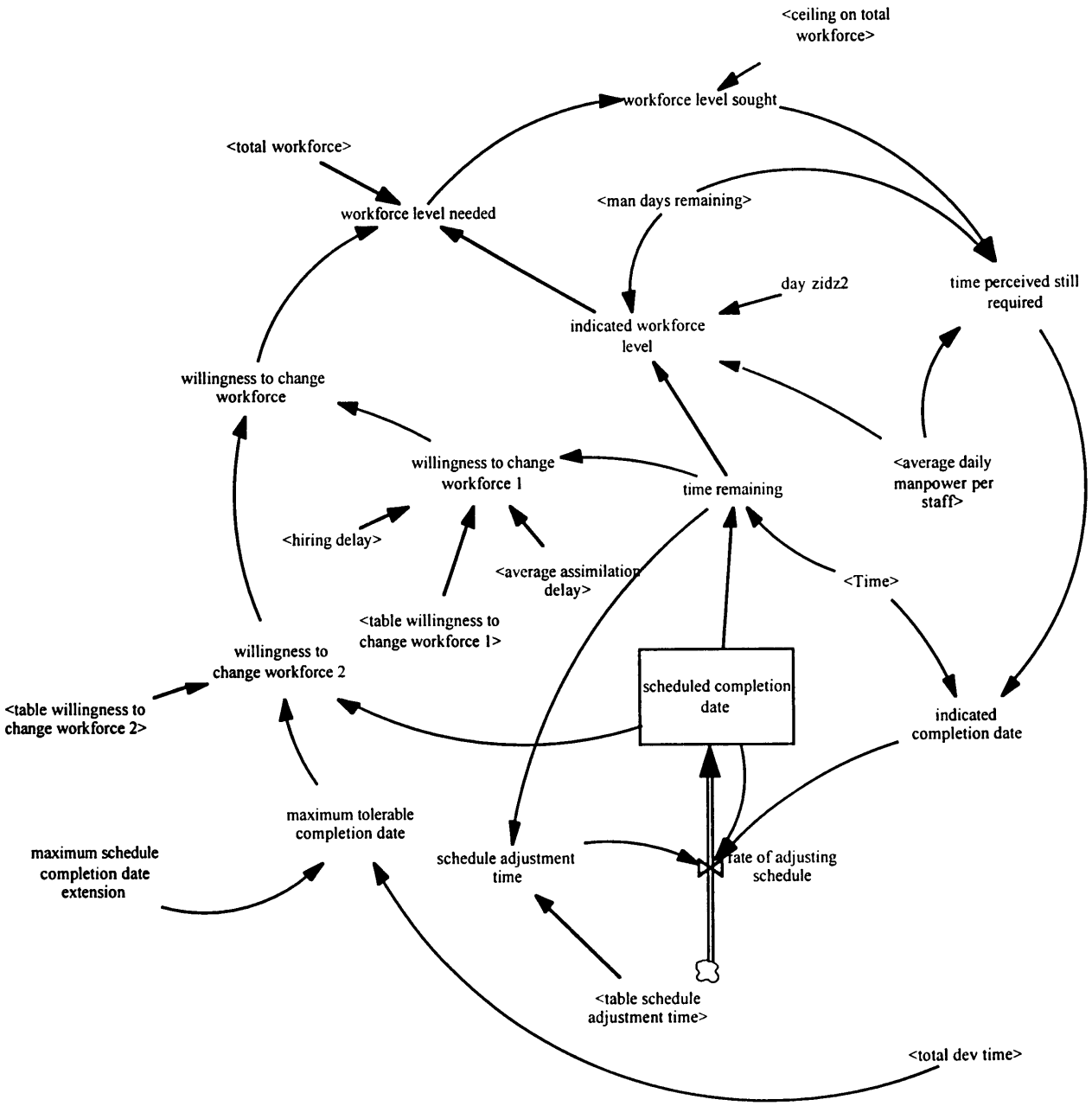

<currently perceived job size in tasks>
<daily manpower for dev and testing>
fraction of effort for system testing
daily manpower for software dev
<TIME STEP>
<Time>
software dev rate
<tasks perceived remaining>
<table fraction of effort for system testing>
tasks dev
<software dev productivity>
QA rate
average QA delay

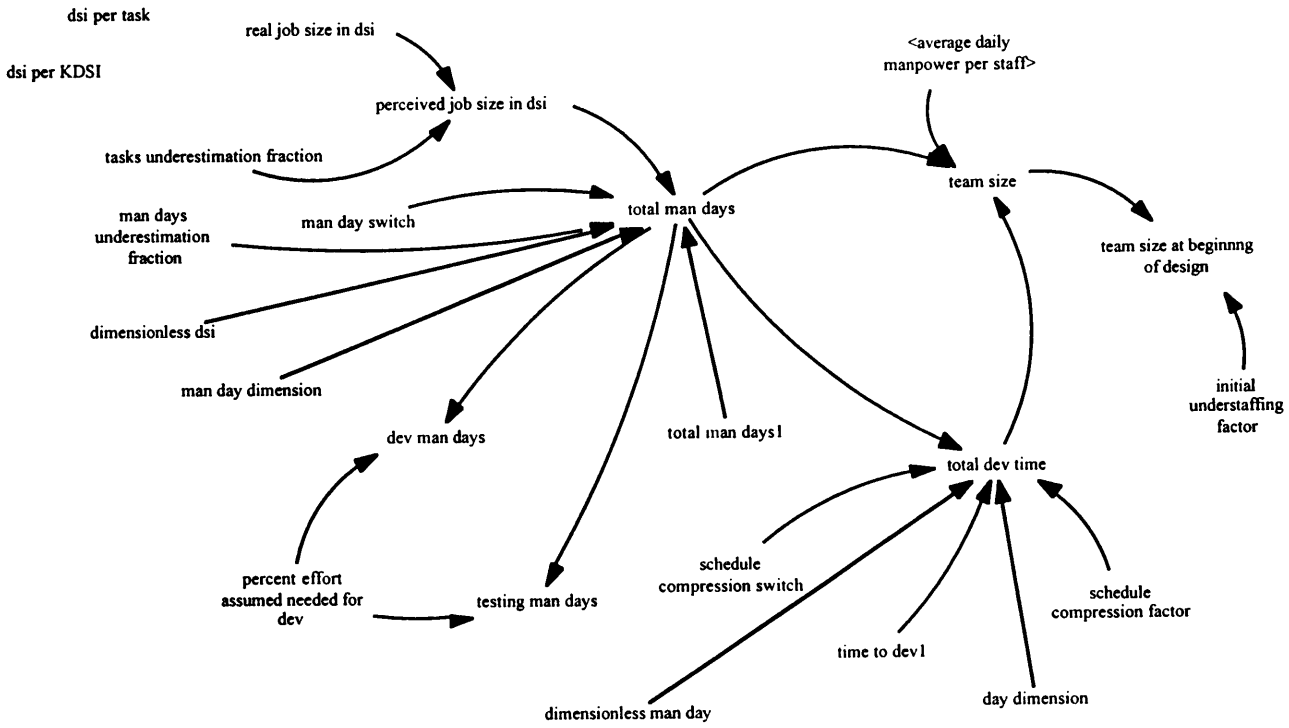

<experienced workforce>
<table communication overhead>
exhaustion depletion delay time
nominal potential productivity new
<total workforce>
nominal potential productivity experienced
fraction of experienced workforce
average nominal potential productivity
communication overhead
rate of depletion in exhaustion level
potential productivity
multiplier to productivity due to motivation and communication losses
nominal fraction of man days on project
exhaustion
software dev productivity
<percent of job actually worked>
multiplier to potential productivity due to learning
effect of work rate sought
actual fraction of man day on project
rate of increase in exhaustion level
<table multiplier to potential productivity due to learning>
perceived excess or shortage in man days
work rate adjustment delay
work rate adjustment rate
<table rate of increase in exhaustion level>
<man days remaining>
<nominal fraction of man days on project>
<total man days perceieved still needed>
percent boost in work rate sought
work rate sought
normal work rate adjustment delay
effect of exhaustion on overwork duration threshold
days zidz3
<man days zidz3>
control switch
handled man days
<table normal work rate adjustment delay>
<time remaining>
<full time equivalent workforce>
<perceived excess or shortage in man days>
nominal overwork duration threshold
<table effect of exhaustion on overwork duration threshold>
man day excesses that will be absorbed
maximum man days shortage handled
overwork duration threshold
<table nominal overwork duration threshold>
<time remaining>
<table man day excesses that will be absorbed>
<exhaustion>
maximum tolerable exhaustion
<man days remaining>
maximum boost in man hours
exhaust inflow
willingness to overwork
exhaust switch
<Time>
variable that controls time to de exhaust
exhaust variable flush
time of last exhaustion breakdown
Input time of last breakdown
breakdown flush
<TIME STEP>
<overwork duration threshold>
<TIME STEP>
breakdown switch


<table multiplier to active error generation due to error density>
<dsi per task>
multiplier to active error generation due to error density
time to smooth active error density
<percent of job actually worked>
<table fraction of escaping errors that will become active>
<software dev rate>
active error regeneration rate
active error density
tasks zidz
fraction of escaping errors that will become active
<error escape rate>
bad fixes generation rate
<rework rate>
percent bad fixes
active error generation rate
undetected active errors
active detection and correction rate
<TIME STEP>
active error retiring fraction
<percent of job actually worked>
<table active error retirement fraction>
active error retirement rate
cumulative errors reworked in testing phase
<cumulative tasks QAed>
<TIME STEP>
cumulative testing man days
undetected passive errors
passive error density
passive error generation rate
testing rate
<tasks zidz3>
passive detection and correction rate
daily manpower for testing
<TIME STEP>
cumulative tasks tested
<fraction of effort for system testing>
testing manpower needed per task
dimension factor
testing manpower needed per error
testing overhead
<daily manpower for dev and testing>
<cumulative errors reworked in testing phase>
<undetected active errors>
all errors that escaped and were generated
<active error density>
<dsi per task>
<multiplier to productivity due to motivation and communication losses>
<passive error density>
percent of tasks tested
<undetected passive errors>
all errors reworked in dev and testing
<currently perceived job size in tasks>
<detected errors>
all errors
<cumulative reworked errors during development>
<potentially detectable errors>


<rate of increase in testing due to discovered tasks>
<table multiplier to productivity weight due to resource expenditure>
<total job size in man days>
planned testing size in man days
<fraction of effort for system testing>
<rate of adjusting job size in man days>
<TIME STEP>
multiplier to productivity weight due to resource expenditure
<detected errors>
<perceived rework manpower needed per error>
man days perceived needed to rework detected errors
man days perceived remaining for new tasks
multiplier to productivity weight due to dev
<table multiplier to productivity weight due to dev>
<percent of perceived job dev>
man days zidz
projected dev productivity
<tasks perceived remaining>
weight of projected productivity
<man days remaining>
<cumulative man days expended>
assumed dev productivity
man days reported still needed
schedule pressure
reported shortage excess in man days
<cumulative testing man days>
perceived dev productivity
total man days perceived still needed for new tasks
<perceived excess or shortage in man days>
<handled man days>
<cumulative tasks dev>
total man days perceieved still needed
<real job size in tasks>
man days perceived still needed for testing
time to smooth testing productivity
percent of job actually worked
<currently perceived job size in tasks>
perceived testing productivity
cumulative tasks dev
tasks remaining to be tested
actual testing productivity
<cumulative testing man days>
<software dev rate>
planned testing productivity
<planned testing size in man days>
<cumulative tasks tested>
man days zidz2


<dsi per task>
<real job size in dsi>
real job size in tasks
<table percent of undiscovered tasks discovered per day>
undiscovered job tasks
percent of undiscovered tasks discovered per day
rate of discovering tasks
tasks discovered
average delay in incorporating discovered tasks
percent of perceived job dev
rate of incorporating discovered tasks into project
<cumulative tasks dev>
currently perceived job size in tasks
tasks perceived remaining
man days remaining
<cumulative man days expended>
<man days reported still needed>
rate of adjusting job size in man days
<table delay in adjusting job size in man days>
<time remaining>
delay in adjusting job size in man days
total job size in man days
<assumed dev productivity>
rate of increase in dev man days due to discovered tasks
<perceived testing productivity>
rate of increase in testing due to discovered tasks
fraction of additional tasks added to man days
<table fraction of additional tasks added to man days>
perceived size of discovered tasks in man days
<assumed dev productivity>
<tasks discovered>
relative size of discovered tasks
<man days perceived remaining for new tasks>
man days zidz3
maximum relative size of additions tolerated without adding to project man days
<Time>
<man days reported still needed>
<total job size in man days>
percent of task reported complete
reporting delay
<man days perceived still needed for testing>
<planned testing size in man days>
percent of dev perceived complete


<ceiling on total workforce>
workforce level sought
<total workforce>
workforce level needed
<man days remaining>
day zidz2
time perceived still required
indicated workforce level
willingness to change workforce
willingness to change workforce 1
time remaining
<average daily manpower per staff>
<hiring delay>
<average assimilation delay>
<Time>
<table willingness to change workforce 1>
willingness to change workforce 2
scheduled completion date
indicated completion date
<table willingness to change workforce 2>
maximum tolerable completion date
maximum schedule completion date extension
schedule adjustment time
rate of adjusting schedule
<table schedule adjustment time>
<total dev time>


dsi per task
real job size in dsi
dsi per KDSI
perceived job size in dsi
<average daily
manpower per staff>
tasks underestimation fraction
team size
man days
underestimation
fraction
man day switch
total man days
team size at beginnng
of design
dimensionless dsi
man day dimension
initial
understaffing
factor
dev man days
total man days1
total dev time
schedule
compression switch
percent effort
assumed needed for
dev
testing man days
schedule
compression factor
time to dev1
dimensionless man day
day dimension

Appendix C:

ACE Specifications for PERIOD1

Efficiency

Thresholds

0% <= efficiency <= 150%

Base efficiency

Normal: Skill level of person/Skill level of task
Outside Resource: 0 % for first period on project. AN outside person is one who has never worked on the phase before
Task dependency - If an individual is working on a task that is dependent on an unfinished task, then that person's efficiency is 15%

Deductions (only applied to "normal" case from above, all other "base" efficiencies stand alone)

10% deduction when internal resources start a task that is already underway (> period 1)

5% deductions are taken for each additional task that a resource is working on. For example, 1 task assigned = no deduction, 2 tasks assigned = 5% efficiency deduction on each. Clarification, the coefficient, 5 in this case, is multiplied by the number of tasks over 1. So 1 task = no penalty, 2 tasks = 5% penalty on each, 3 tasks = 10% penalty on each

Overtime deductions will be taken based solely on the number of assigned hours in that period.

Morale deductions/additions are pulled directly from the current morale score. A morale score of 2 means add 2% to efficiency, a morale score of -2 means subtract 2% efficiency

20% deduction if the tasks scope creep flag is set to true

Skill-building efficiency deductions (these deductions kick-in when the resource is engaged in either knowledge capital or coaching mode

There is no rework coefficient in this system

MORALE

The morale score will take a value between -10 and 10. It is based on five variables. The following table summarizes how to compute the morale score. For each variable there are two rows in the table. The first row contains the value that should be added or subtracted to the

morale score. The second row contains the lookup for the current value of the variable. The computation to determine the current value of each variable is explained elsewhere. For each period, morale starts at 0 and has these numbers applied.

Communication efficiency	-2	0	2
	<=40%	>40%, <80%	>=80%

Skill match	-2	0	2
	>= 2	>=1,<2	<1

OT trend	-2	0	2
	>=25%	<25%,>10%	<=10%

Team Morale	-1	0	1
	<= 3	>3, <7	>=7

Development need being met	-3	0	3
	>=-1, <-.33	>=-.33, <= .33	> .33, <=1

COMMUNICATION EFFICIENCY

Certain actions that the user takes in one part of the system will have a complimentary "message" that the manager should tell his or her team. The communication score indicates how well the manager follows through on his or her actions by communicating to the team.

Communication efficiency = # of communications for actions/# of actions requiring communication

NOTE: Both of the variables that make up communication score are reset to zero at the beginning of each period. (i.e., the manager only gets "credit" if he or she communicates an action prior to hitting the "run" button.

Implications

For each action that can be taken (e.g., making a change to the start date, assigning resources, designing resources, etc.) we (the developers) need to be able to designated whether it is an action which requires communication to the team and associate some type of message ("I changed the start and end dates in the workplan, please take a look"). We need to identify how detailed these messages will be. For example, if we just say ("I changed our schedule, please review the workplan"), then there would only need to be one statement regardless of the number of tasks for which we changed

schedule. If on the other hand, we want it to say, "I changed the schedule for task XXX", then each change will have to have its own communication. For simplicity, I am tending to lean toward the former. Any thoughts?

When the user takes an action that needs to be communicated, the associated message needs to be added to the team conversation tool as something the user can say (this happens within the period, not after the run).

We need to keep a variable around that counts the number of actions that require communication. This number should be an aggregate of all actions requiring communication. The following table summarizes some considerations for updating and tracking this value.

Action sequence	# of items requiring communication
User makes change to action that requires comm	1
User make change to action not requiring comm	0
User makes change to action requiring comm, then makes second change to same action that does not set it back to original value	1
User makes change to action requiring comm, then makes second change to same action that sets it back to original value	0
User makes change, communicates to team and then makes another change to same action	1 after first change 0 after communication 0 after second change (even if it resets action)

One way to handle this is as follows:

Keep a queue of actions and their original values. When a user makes an action that requires comm, the queue is checked. If the action is not in the queue it gets added, if the action is in the queue but takes a different value (not its original value) then nothing happens, if the action is in the queue and it takes its original values, the action is removed from the queue. In addition, if a user makes the statement associated with the action it is also removed from the queue and a variable tracking # of communicated actions gets incremented.. Then when the simulation is run, the total number of actions requiring communication will be the total number of communicated actions plus the length of the queue.

We need to have a variable that count the number of actions that have been communicated (see prior paragraph for ideas on how to update)

We'll need to have a property associated with statements in the tree which indicate what action (if any they are communicating)

SKILL MATCH

skill match = | person's skill - task skill |

In cases of multiple tasks for an individual, the match should be the average across tasks for the individual

OT TREND

$(\sum_{p}^{p-2} OT) / 3$, where p = current period

Team morale

Average team morale from prior period

Development need

-1	0	1
Current task is one I want to avoid	Current task is one for which I am indifferent	Current task is one that I want to pursue

For each individual we will need to have two properties: tasks-I-want-to-avoid and tasks-I-want-to-pursue. Any task which is not in either list can be considered indifferent. If team member is on multiple tasks then use the average.

Developers: Information regarding the tasks I want to pursue and avoid should come out through team member biographies and team conversation. In the bio, it should be indicated as "development areas" or "areas for improvement" like from a CMAP form. In conversation it should come out as, "I'd really like to do X" or "I've done a lot of Y, I'd like to move on". In general, 90% of tasks should be indifferent with 5% falling into each property of want-to or avoid.

RESET THRESHOLD

Users will be reset to the beginning of a scenario under any of the following conditions

SQERT Meter

3 red T's in a row
3 red E's in a row

Trend meters

3 extreme CV downwards
3 extreme SV downwards

Absolute numbers

Ledger actual exceeding budget by x%
Current date exceeding milestone date by x%

The reset decision should be made after all of the other engine computations are made. At that point if the thresholds are exceeded the user should be reset. The only exception to this rule is if the user has completed all tasks but has exceeded the thresholds they should NOT be reset. The only reset criteria which should be impacted by this are the absolute numbers. It would be highly unlikely that the phase could be finished on the third red or downward trend.

Scope creep

The scope creep flag is an indicator that the current task is creeping in scope. When it is turned on, the resources assigned to the task will lose an additional 20% efficiency each. The scope creep flag needs to turned on/off via the simulation. When the flag is turned on any other tasks and/or substasks within the same competency strand that are open or not started will also have their flags turned on. Conversely, turning off the flag at point should turn off all of the other flags in a similar manner.

There is no scope creep threshold.

Overtime

Minimum and maximum overtime thresholds

In the workplan, the user will be able to identify project-level thresholds for maximum allowable and minimum required. The thresholds will be applied as follows:

Maximum allowable OT is used when determining self-induced overtime. SI overtime cannot exceed maximum allowable, however, OT assigned to a task overrides if it is greater

Minimum OT – this is the lowest value that Self induced OT can take Task assigned OT overrides minimum OT.

Self induced overtime

Self induced overtime is intended to simulate the fact that people will tend to work mode as deadlines approach or as they fall behind. Self induced overtime is computed as follows:

Self induced overtime =

Greater of:

MinSI

or

Self Induced OT Factor * MaxAllowableSI (which defaults to 50% but can be set by the user)

The following table is used to compute the self-induced OT Factor:

		Percentage of time into task				
		0-.25	.26 - .5	.51 - .75	.76 - 1	1.01 +
SPI (BCWP/ BCWS)	0-.25	.4	.6	.8	1	1
	.26 - .50	.3	.4	.6	.8	1
	.51 - .75	.2	3	.4	.6	1
	.76 - .99	.1	.2	.3	.4	1
	1+	0	0	0	0	0