Modeling dynamic interactions in a software development project

Chetan D. Vajre
University of South Florida

Follow this and additional works at: http://scholarcommons.usf.edu/etd

Part of the American Studies Commons

Scholar Commons Citation

This Thesis is brought to you for free and open access by the Graduate School at Scholar Commons. It has been accepted for inclusion in Graduate Theses and Dissertations by an authorized administrator of Scholar Commons. For more information, please contact scholarcommons@usf.edu.
MODELING DYNAMIC INTERACTIONS IN A SOFTWARE DEVELOPMENT PROJECT

by

CHETAN D. VAJRE

A thesis submitted in partial fulfillment of the requirements for the degree of Master of Science in Industrial Engineering Department of Industrial and Management Systems Engineering College of Engineering University of South Florida

Major Professor: Suresh K. Khator, Ph.D.
William Miller, Ph.D.
Ali Yalcin, Ph.D.

Date of Approval:
April 4, 2003

Keywords: system dynamics, simulation, vensim, waterfall model, project management
© Copyright 2003 , Chetan D. Vajre
# TABLE OF CONTENTS

**LIST OF FIGURES**

<table>
<thead>
<tr>
<th>Chapter</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ABSTRACT</td>
<td>v</td>
</tr>
</tbody>
</table>

**CHAPTER 1 INTRODUCTION**

1.1 Introduction to Business Scenario 1

1.2 Simulation of Existing Systems 2

1.3 Need for Simulation 3

1.4 A System Dynamics Modeling Approach 4

1.5 Modeling Using Vensim 6

1.6 Organization of the Thesis 12

**CHAPTER 2 LITERATURE REVIEW**

2.1 Introduction to Software Project 13

2.2 Basic Definitions of Project Management 14

2.3 Early Project Management Techniques 15

2.3.1 Gantt Chart 15

2.3.2 Critical Path Method 16

2.3.3 Program Evaluation and Review Technique 16

2.4 Recent Project Management Methods 18

2.4.1 AND-OR Graphs 18

2.4.2 Petri-Net 18

2.4.3 Limitations AND-OR and Petri-Net Graphs 20

2.6 V Model of Software Development 23

2.7 System Dynamics Modeling of Software Project 27

**CHAPTER 3 PROBLEM STATEMENT**

3.1 Brook’s Law 30

3.2 Integrative System Dynamics Model of Software Development 31

3.3 Factors Under Consideration 31

3.3.1 Effort Required 32

3.3.2 Time in Days 32

3.3.3 Staff Levels 32

3.3.4 Productivity 33

3.3.5 Testing and Error Detection 33

3.4 Problem Description and Objectives 34

3.4.1 Performance Measures 34
LIST OF FIGURES

<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Example of an Inventory –Workforce Model in Vensim</td>
<td>7</td>
</tr>
<tr>
<td>2</td>
<td>System Dynamics Model of Rabbit Population</td>
<td>9</td>
</tr>
<tr>
<td>3</td>
<td>Plot of Rabbit Population versus Time</td>
<td>10</td>
</tr>
<tr>
<td>4</td>
<td>Using Synthesim Feature of Vensim</td>
<td>11</td>
</tr>
<tr>
<td>5</td>
<td>Causes Strip Plots</td>
<td>11</td>
</tr>
<tr>
<td>6</td>
<td>Work Breakdown Structure</td>
<td>14</td>
</tr>
<tr>
<td>7</td>
<td>A Typical Gantt Chart</td>
<td>15</td>
</tr>
<tr>
<td>8</td>
<td>Beta Distribution for PERT Activities</td>
<td>17</td>
</tr>
<tr>
<td>9</td>
<td>A Sender Receiver Petri Net</td>
<td>19</td>
</tr>
<tr>
<td>10</td>
<td>Waterfall Model of Software Development</td>
<td>22</td>
</tr>
<tr>
<td>11</td>
<td>V Model of Software Development</td>
<td>25</td>
</tr>
<tr>
<td>12</td>
<td>A Cause-Effect Relationship</td>
<td>27</td>
</tr>
<tr>
<td>13</td>
<td>Feedback Loops</td>
<td>28</td>
</tr>
<tr>
<td>14</td>
<td>Workforce Sub-System</td>
<td>37</td>
</tr>
<tr>
<td>15</td>
<td>Productivity Sub-Systems</td>
<td>39</td>
</tr>
<tr>
<td>16</td>
<td>Lookup Graph for Productivity</td>
<td>39</td>
</tr>
<tr>
<td>17</td>
<td>Lookup Graph for Time to Detect Errors</td>
<td>40</td>
</tr>
<tr>
<td>18</td>
<td>Work Rate Sub-System</td>
<td>40</td>
</tr>
<tr>
<td>19</td>
<td>Cost Sub-System</td>
<td>41</td>
</tr>
<tr>
<td>Figure</td>
<td>Description</td>
<td>Page</td>
</tr>
<tr>
<td>--------</td>
<td>-------------</td>
<td>------</td>
</tr>
<tr>
<td>20</td>
<td>Work Accomplished</td>
<td>45</td>
</tr>
<tr>
<td>21</td>
<td>Total Cost</td>
<td>46</td>
</tr>
<tr>
<td>22</td>
<td>Work Flow</td>
<td>47</td>
</tr>
<tr>
<td>23</td>
<td>Feedback Loop</td>
<td>50</td>
</tr>
<tr>
<td>24</td>
<td>Lookup Graph for Pressure</td>
<td>54</td>
</tr>
<tr>
<td>25</td>
<td>Tabular Data of Project is Done Variable</td>
<td>56</td>
</tr>
<tr>
<td>26</td>
<td>Model Run in Synthesim Mode</td>
<td>56</td>
</tr>
<tr>
<td>27</td>
<td>Feedback Model in Game Mode with Ten-Month Deadline</td>
<td>58</td>
</tr>
<tr>
<td>28</td>
<td>Feedback Model in Game Mode with Fourteen-Month Deadline</td>
<td>59</td>
</tr>
<tr>
<td>29</td>
<td>Hiring Rate with an Eight-Month Deadline</td>
<td>60</td>
</tr>
<tr>
<td>30</td>
<td>Project State at the End of Five Months</td>
<td>62</td>
</tr>
<tr>
<td>31</td>
<td>Custom Graphs I</td>
<td>65</td>
</tr>
<tr>
<td>32</td>
<td>Custom Graphs II</td>
<td>66</td>
</tr>
<tr>
<td>33</td>
<td>Welcome Screen</td>
<td>67</td>
</tr>
<tr>
<td>34</td>
<td>Control Panel</td>
<td>67</td>
</tr>
<tr>
<td>35</td>
<td>Instructions View</td>
<td>68</td>
</tr>
<tr>
<td>36</td>
<td>Gaming Controls in Vensim</td>
<td>70</td>
</tr>
</tbody>
</table>
MODELING DYNAMIC INTERACTIONS IN A SOFTWARE DEVELOPMENT PROJECT

Chetan D. Vajre

ABSTRACT

Software industry is getting very competitive in the wake of recession. In most cases, an organization that quotes a lower price and promises to deliver the product at the earliest walks away with the project. But the factor of quality of the product delivered is also very important because that in turn determines the reputation of an organization, which also plays an important role in getting the next project.

Interactions in a software development are dynamic in nature and involve human factors. Models are built taking into account all possible factors so as to present a realistic picture of the development process. System dynamics methodology is used to build these models in Vensim. Three models have been proposed to help manager estimate an approximate time and cost of the project, monitor the project once the timeline is set and monitor the project development to change various factors as the development process goes through various phases of development and testing.
CHAPTER 1
INTRODUCTION

1.1 Introduction to Business Scenario

There has never been a perfect recipe for the success of a business. There are a very few who get the perfect combination in place for a successful business. There might be some subtle differences in running the same business, manufacturing the same product or for instance developing very similar software that separates the top notch from the rest. The one who gets things done earliest and at minimum cost with an acceptable quality will survive in this competitive business world.

Whenever there is a failure of a business, or software developed fails, or a product does not do what it says it does, people make the cardinal mistake of looking into individuals responsible for a particular phase in manufacturing. The reason might well be something else. For example, a software product is developed and it does not sell in the market. It may perform well, but maybe because the client or the people do not use the platform on which the product, it does not do well in the market. So it doesn’t help making scapegoat of the developers or the testers who tested the software. We now need to develop a futuristic view of the arena around us and start predicting what the future needs might be and what would be available at hand. Sometimes a programmer does not introduce bugs because he is not good at programming, the case might well be that
fatigue has started affecting his work. With break-even status hard to achieve and economy experiencing its worst phase ever, companies and organizations have started looking towards new and advanced avenues to help them earn profits. The focus has now shifted from looking at individual performances to the performance of the system as a whole. Managers have now started believing that some intangible factors may also be affecting the performance of the department he leads and so the system output as a whole. For instance reasons for failure might well be the management system, system definition, flaws getting detected at a stage much later than they should be and so on.

Technology has advanced in leaps and bounds and we are presented with the most advanced tools we can ever imagine to help learn and improve. This is more relevant at this time of recession where every competitor is looking to get even the slightest edge. We should therefore seek the most appropriate business practices in order to prevail. It is easier said than done.

It is not easy predicting the future of a business or what changes in a particular aspect might affect the performance. Sometimes we might not be able to afford to experiment, because the failure would be more dreadful than what would have been, had the pattern of work not changed. There is therefore the need for using a surrogate to see what would have been the state of the system had we changed certain factors. This can be achieved by simulation.

1.2 Simulation of Existing Systems

Simulation is a very generic term and applies to a very wide arena and industrial applications. It refers to a broad collection of methods and applications to mimic the
behavior of real systems using a computer with appropriate software. One such widely used software is Arena 5.0 (Rockwell Software).

1.3 Need for Simulation

We can use various theoretical techniques, for example queuing theory, to determine the performance of the system. But sometimes, subjecting the system to different scenarios maybe very repetitive and for large systems may consume lot of time and energy. Instead, in this age, where computing time has become cheap and easily accessible, more and more custom software are available in market to do all that could just have been imagined earlier. There are some very good reasons why we should go for computer simulation:

- There might be only a few limited distributions that we could calculate the results using theoretical derivations.
- The queues may not follow standard rules (in case of system involving queues).
- The system behavior might be too dynamic in nature.
- If we are interested in knowing when the system reaches a steady state, it may be necessary to go on calculating for long time intervals, which is not feasible and practical.
- If we want to put the system through different scenarios, then everything needs to be redone with the new parameters.

But all this looks good when it comes to the modeling of systems where we want to know what the resource utilizations are, what the queue sizes are, etc. However, there are certain systems where such a technique might not be relevant. Consider for example a software industry where the product is new software. Here there are no queues and it’s
difficult to measure utilization of a resource. Also when we want to model and study the behavior of such a system, there are certain factors that might not at first seem to affect the system at all or we can say there are a couple of “intangible” factors that add to the quality of the product. These factors do not add value to the product but if neglected, we might well be off the track of what we expected. A very good example is if we predict a deadline for new software, then absence of a programmer does not actually play a role in the value of the product, but it does affect the deadline. So what is needed is to look at the system as a whole and not just concentrating on the parts that appear to add value to the product. With deadlines and quality becoming a major factor in success of a software industry, more and more importance was being given to start studying the behavior of the system without overseeing even the minuscule. With that view in mind, the field of system dynamics started gaining importance in this most lucrative industry.

1.4 A System Dynamics Modeling Approach

System Dynamics is a methodology for analyzing complex systems and problems with the aid of computer simulation software. Dr J. W. Forrester formulated this methodology in 1960 at M.I.T. He was then a professor at M.I.T’s Sloan school of Management. He became interested in complexity of business management and the causes attributed to its success and failures. He thought that people did not analyze a complex system very well and neglected many factors that indirectly but significantly affect the running of a business. The cardinal mistake committed is attributing the success and failures to some one or two factors. The result is the policy for achieving a goal turns out to be very simple and instead of the system or business behaving as per the expected
pattern, behaves maybe in an exact opposite way leading to the failure of business. For example, lowering the price of the product may not necessarily increase the sales because we may not have taken into consideration the seasonal influence and also the competitor’s new price.

To help facilitate more appropriate decision-making and policies formulation, J.W. Forrester used this concept to include all of the cause-effect relationships, time delays, and feedback loops in the systems. With traditional simulations, we were not able to model cause-effect relationships. So this added a new dimension to modeling techniques. To facilitate such modeling, DYNAMO (Pugh, 1983), a mainframe computer program, was developed. Using dynamo it was possible to investigate the reasons for failures of businesses and also subject it to different scenarios, which are sometimes abstract, for example, taste changes of people, seasonal effects on product sales, etc. Modeling with DYNAMO pointed towards above-mentioned reasons, which we never thought were instrumental in the fluctuations in a business. He then broadened the horizon as to where this methodology could be applied. These included economics, education, time estimations, physical and biological sciences, social issues, etc.

With passage of time, more and more software packages were developed to facilitate this method of modeling. The most popular among them are Vensim, Stella, Powersim, Ithink, etc.

Systems Dynamics models (Coyle, 1996) are more concerned with capturing the structure and policies of the system and with the mode of behavior of the whole system rather than accurate prediction. It is considered that the shape of relationships is more important than their absolute statistical accuracy and that in any holistic approach
accuracy must often be sacrificed in order to remain problem oriented. Once a system
dynamics model has been constructed and the initial conditions are specified a computer
can simulate the behavior of different model variables over time. Computer simulation is
not only useful for modeling systems that are complex in nature but also serve as a
powerful tool influencing the learning process when combined with real experimentation.
Vensim software developed by Ventana Systems, Inc. will be used for developing system
dynamics based models in this thesis.

1.5 Modeling Using Vensim

Vensim is a visual modeling tool that allows the user to conceptualize, document,
simulate, analyze, and optimize models of dynamic systems. Vensim provides a simple
and flexible way of building simulation models from causal loop or stock and flow
diagrams.

By connecting words with arrows, relationships among system variables are
entered and recorded as causal connections. This information is used by the Equation
Editor to help us form a complete simulation model. The model can be analyzed
throughout the building process, looking at the causes and uses of a variable, and also at
the loops involving the variable. Once the model is built and it can be simulated, we can
explore the behavior of the model.

Vensim deals with the following main objects of dynamic simulation

*Levels:* Levels are also known as stocks, accumulations, or state variables.
Levels change their values by accumulating or integrating rates. This means that the
values of Levels change continuously over time even when the rates are changing
For example, in Figure 1 Inventory and Workforce represent levels.

*Flow or Rate:* Rates, also known as flows, change the value of levels. The value of a rate is not dependent on previous values of that rate; instead the levels in a system, along with exogenous. For example, in Figure 1, net hire rate and production rate represents rates.

*Auxiliaries:* Intermediate concepts or calculations are known as auxiliaries and, like rates, can change immediately in response to changes in levels or exogenous influences.

When constructing a Level and Rate diagram, the variables that accumulate over a period of time must be considered. Another way to think about this: if Time slowed discontinuously.

![Figure 1. Example of an Inventory –Workforce Model in Vensim (Ventana Systems, Inc)](image)

For example, in Figure 1 Inventory and Workforce represent levels.

*Flow or Rate:* Rates, also known as flows, change the value of levels. The value of a rate is not dependent on previous values of that rate; instead the levels in a system, along with exogenous. For example, in Figure 1, net hire rate and production rate represents rates.

*Auxiliaries:* Intermediate concepts or calculations are known as auxiliaries and, like rates, can change immediately in response to changes in levels or exogenous influences.

When constructing a Level and Rate diagram, the variables that accumulate over a period of time must be considered. Another way to think about this: if Time slowed discontinuously.
down to zero for the system, what variables would still be nonzero? For example, in the system where we pour water into a glass, the water contained in the glass is the Level. If time froze, the pouring (a Rate) would stop, but we would still see a quantity of water in the glass (a Level). Once we know what levels we need, enter them first and then connect the rates and auxiliaries.

**Sink / Source:** The use of a cloud represents either the source, or the sink of a flow, which is outside the bounds of what we are interested in.

**Equations:** The relationships among the variables are designated by Equations.

- The following steps are typical for building and using Vensim models.
- Construct a model or open an existing model.
- Examine the structure using the structural Analysis tools (Tree Diagrams).
- Simulate the model moving around model parameters to see how it responds.
- Examine interesting behavior in more detail using the dataset Analysis tools.
- Perform controlled simulation experiments and refine the model.
- Present the model and its behavior to the audience using Synthesim results.

Let us take a simple example of a population behavior in rabbits. Here the Level/Stock is the present population. This is affected by births and deaths. Now births tend to increase it and deaths decrease. So we sketch the system as shown in Figure 2.
Now we need to write equations using the equation tool. We enter the following equations:

Rabbit Population = births - deaths

Births = birth rate * Rabbit Population

Deaths = Rabbit Population/average lifetime

Average Lifetime = 8 (Constant)

birth rate = 0.125 (Constant)

Rabbit Population = 1000 (Constant)

With these set of parameters, we now run the model and say the first run we call it as the ‘equilib’ and click simulate. Since both birth rates and death rates are equal, when we run the model and use the graph tool by choosing ‘Rabbit Population’ as the “Benchwork” variable, we find that it’s a straight line that coincides the 1000 rabbit population maximum value.
A key feature of Vensim is its ability to do multiple simulations on a model under different conditions to test the impact that changes in constants (or lookups) have on model behavior. Vensim also stores all the data for all variables for each simulation run, so that we can easily access information about the behavior of any variable in any run. Temporarily changing constant or lookup values and then simulating the model experiments are performed. For this the SyntheSim feature is used.

It is seen that by just moving the slider for the various variables, Vensim automatically plots the changes. This is a very powerful feature of Vensim. The ‘birth rate is changed to 2 instead of 0.125. Immediately if the mouse is moved over the Rabbit Population, another window pops up showing the population growing exponentially. The light lines show the current run and the dark line the \textit{equilib} run results. If we click the “causes strip” option in the toolbar menu, we get Figure 5.

**Figure 3. Plot of Rabbit Population versus Time (Ventana Systems, Inc)**
The causes strip feature is particularly useful in tracing the exact cause of the behavior. By plotting each benchwork variable against its predecessor or the one that causes it, we move backwards and try to trace where the deviation comes from. For example, in Figure 1 on page 7, if we use the causes strip tool, we come to know that even tough inventory is changed by both sales and production, only production has oscillating behavior and so we must look into production and not sales as we have an oscillating behavior for inventory.

Thus we see how we can model systems, which were initially not feasible, as we did not have relevant objects in the earlier software used to model systems. We will be using this tool to model our system under study.

Figure 5. Causes Strip Plots
1.6 Organization of the Thesis

This chapter dealt with the reason for modeling software development projects using system dynamics. Also it gave a brief description of the tool that we are going to use to model and some basic features and analysis tools of Vensim. The thesis is organized in six chapters. The literature review chapter looks into all the work that is done in the field of software project modeling using system dynamics. In chapter three we define our problem statement. The emphasis would be on the staffing requirements and the effect of changes in the level of experience of the staff on the project duration and cost. Then a list of assumptions that were made for model development is given. In Chapter four, initial results are provided. Chapter five deals with building user-friendly interfaces for the models. Finally, chapter six considers incorporating feedback factor in models, conclusion and also what would be some areas that could be studied for any further research.
CHAPTER 2
LITERATURE REVIEW

Now that we know the basics of simulation and system dynamics, we can delve into the topic of project management. We would look into the proposed models for software projects, their utility as well as their shortcomings.

2.1 Introduction to Software Project

A project consists of a series of activities directed to accomplish a desired goal. It includes a minimum set of features including:

- Specific objective to be completed with specified details,
- Start and End dates defined,
- Budget/Cost, and
- Resources (People/Equipments).

A successful software project is the one that meets the customer’s requirements, possesses high quality and is delivered on time and on budget. It is rightly said, “Poor management can increase software costs more rapidly than any other factor.” There are very solid reasons to start thinking of modeling a software project.

- Once modeled, it becomes easy to understand and describe what exactly are the steps to be taken.
- It becomes a common language between the developers and the customers and
between the developers themselves. This reduces communication overhead.

- Milestones can be set and a constant track of the progress can be kept by comparing with the theoretical value.

- It provides a tool for software engineers to build tools that will support and enhance the software projects.

Let us get familiar with some basic terms in project management and system dynamics.

### 2.2 Basic Definitions of Project Management

- An activity is a task with well-defined beginning and well-defined end that can be performed by single functional entity.

- An event (milestone) is an occurrence at a point in time that signifies the start or completion of one or more task or activities.

- A work breakdown structure (WBS) takes the project and "breaks it down" into the activities, which must be accomplished to complete the project. A WBS in tree form is shown in Figure 6.

- Project management is a mixture of people, resources, systems and techniques required to carry project to successful completion.

![Spreadsheet Project Diagram](image)

Figure 6. Work Breakdown Structure (Horowitz and Lia, 1989)
We would now discuss some of the models of project management that have been proposed for software project management.

### 2.3 Early Project Management Techniques

#### 2.3.1 Gantt Chart

This is a calendar-oriented chart that has lines indicating project activities. A typical Gantt Chart is show in Figure 7. A Gantt chart displays separate events that have a defined starting and ending value. As such, time charts are excellent for planning the use of resources. The scale used for comparison may be specified using dates, time units, or numeric values.

There are special marks indicating milestones or critical activities (those activities that cannot be delayed without delaying the project).

![Figure 7. A Typical Gantt Chart (Figure from Microsoft Visio)](image-url)
2.3.2 Critical Path Method

This approach/method emphasizes the interconnection between activities. The collection of activities and links form acyclic graph. Every activity has a start and end event associated with it. The predecessor and the successor relationships govern the order of activities.

Using Gantt Chart and Critical Path Method, we could find the

- Earliest start time.
- Latest Start time.
- Earliest Ending Time.
- Latest Ending time.
- Slack.

2.3.3 Program Evaluation and Review Technique

This is related to Critical Path Method but is more flexible with start and end times. We have three time estimates of each activity

- Probable earliest completion time
- Probable latest completion time
- Most probable completion time

From the terminology itself, one can guess its more probabilistic approach. Time is measured by random variables with and assumed probability distribution. For example say beta distribution for activity times as shown in Figure 8.

Thus,

Mean time for activity = \((a + 4m + b)/6\)
Variance = \[(b - a)/6\]^2

Figure 8. Beta Distribution for PERT Activities

These seem only theoretical models with more stress on times. There are no considerations of human factors or other factors that may influence these times.

We could easily see that:

- A manager if uses these techniques, he has no tool to analyze and keep a continuous track of the progress of activities.
- It does not represent WBS as integral system component.
- There is no provision for intelligence in system (decision making).
- Without any inherent intelligence incorporated in the system, the system cannot adjust to changing conditions.
- There is no log as to what caused an activity to start (event that fired the activity).

To know the exact reasons for behavior of system and allow decision making or for instance automatic reassignment of responsibilities in case of deviation from planned goal and deadlines, some kind of intelligence needed to be added to the models. With that view in mind, people started modeling such systems using the following two
methodologies:

- AND-OR Graphs
- Petri Nets

We would just overview them and not get into too much details, as they are very separate fields in themselves.

2.4 Recent Project Management Methods

2.4.1 AND-OR Graphs

An AND-OR Graph (Horowitz and Liu, 1989) is a directed acyclic graph and has three types of nodes

- A AND node denotes an object that is the aggregation of all of its predecessor nodes.
- An OR node denotes an object defined by only one of its predecessor nodes.
- LEAF nodes denote atomic entities (An entity that cannot be divided is called an atomic entity). They have no outgoing arcs.

AND-OR graphs did add to the modeling abilities by introducing decision making at steps but it anyways did not consider human factors. So, though it was an improvement over the earlier models, the change was not significant enough for people to accept it completely.

2.4.2 Petri-Net

This is an abstract model for describing and analyzing information and control flow in asynchronous concurrent systems. A typical Petri-net of a sender receiver system is shown in Figure 9 (Peterson, 1977). This is the good example to understand a Petri net. As we can see, it has places (represented by circles) where one or more tokens
(represented by small dots) reside. The straight lines are the transitions that are fired when all the input places have at least a token in them. The tokens flow along the directed arrows.

Formally a Petri Net $C$ is defined as a four-tuple

$$C = (P, T, I, O)$$

Where,

$P =$ set of places

$T =$ set of transitions

$I =$ set of input places

$O =$ set of output places

**Figure 9. A Sender Receiver Petri Net (Peterson, 1977)**
Directed arcs from places to transitions and from the transition to places are represented by Input-Output functions. As we can see from the Figure, the sender is in a processing state (i.e. formatting a message) the receiver is in ready-to-receive state waiting for any incoming message. When the message is transmitted, the send message transition is fired and two tokens are created, one to wait for response place on sender side and the other to the message-send place on the medium. Now the receiving message transition is enabled and the receiver transits to message received state. After message is verified, receiver sends an acknowledgement response to the sender and starts to process this message. When the sender side receives this response, the sender goes back to processing state and is ready to accept any further request. When the processing of message is finished, the receiver also goes back to ready-to-receive state.

2.4.3 Limitations of AND-OR and Petri-Nets Graphs

Though these models seem to be intelligent in some way and very similar to computer systems, they did not gain popularity because

- AND-OR graphs describe only vertical structures
- They do not provide dependency relationship among different information types (Information exchange between departments)
- There is no track of resources consumed for a particular task.
- However, the application of Petri-Nets to project management had many limitations.
- A token is only a boolean representation. It does not take into account the state of the system that is best described by its attributes.
Execution of Petri nets is non-deterministic. It means if we have two transitions enabled at the same time, one does not know which is going to fire (we do not know the exact cause of an activity) unless we look into the firing sequence and in a large Petri-Net, this could be time consuming.

It is difficult to model human factors using Petri-Nets and AND-OR graphs. Colored Petri nets did show an improvement but not good enough to completely model a project development system. Also a hybrid model called “Design Net” that came much later did not gain popularity.

2.5 Waterfall Model of Software Development

The most popular model, which though one can say does not identify very, truly with the development process, is the “Waterfall Model.” The Waterfall Model (Royce, 1977) of project management is also referred as the system development life cycle model. The system development life cycle model is an approach in development of an information system or software product that is characterized by linear sequence of steps in which we can never go back. It is called the waterfall model because we can compare it to the waterfall on a cliff of a steep mountain. Once the water falls over the cliff edge and begins its journey down, it cannot go up. This is the oldest model.

In this model, there are benchmarks set at the end of every step or phase. Once a phase of development is completed, development proceeds to next phase and there is no turning back. In this model, one cannot go back even if some conditions may demand that. This model though old is still used as a conceptual guideline for almost all of Air Force and NASA software development.

A schematic overview of waterfall model, representing concurrent hardware and
software development is shown in Figure 10. The advantage of waterfall model is that it allows for departmentalization and managerial control. We can set deadlines at each stage and the product moves like an assembly line.

Figure 10. Waterfall Model of Software Development (Royce, 1987)

The development process moves from the following phases:

- User Requirement Analysis,
- System Requirement Analysis,
- Preliminary Design,
- Detailed Design,
Coding and computer software unit (CSU) testing, and

Computer Software component (CSC) Integration and Testing

As we can see, the waterfall model does show the phases of software development as it goes through, but one can see distinctly that we cannot go back. Besides these, below we list the following limitations.

Every situation needs a new model, so one cannot use this for all software projects. This is a very generalized model.

- It does not consider requirement changes.
- End user is never considered in model development.
- Fails to treat software development as a problem solving process and so fails to give insight into actions and events that precede the finished products.
- Does not allow for much reflection and revision.
- We cannot go back from any stage. This is a major disadvantage. Any improvement that needs going back from testing stage cannot be incorporated.
- Keeping these things in mind, the V Model was proposed.

2.6 V Model of Software Development

As an improvement over the waterfall model, in early 1980s it was suggested that instead of taking testing to a new phase altogether, it should be given equal weight and not treat it as an afterthought. This was the most significant change needed cause the faults were detected much later than they should be.

The V model (www.convergsoft.com) was included in the U.K.'s National Computing Centre publications (Convergsoft Systems) in the 1990s with the aim of improving the efficiency and effectiveness of software development. It is accepted in
Europe and the U.K. as a superior alternative to the waterfall model; yet in the U.S., the V Model is often mistaken for the waterfall model.

The V Model portrays several distinct testing levels and illustrates how each level addresses a different stage of the lifecycle. The V shows the development activities on the left-hand (downhill) side and the corresponding sequence of test execution activities on the right-hand (uphill) side.

On the development side, we start by defining business requirements, then successively translate them into high- and low-level designs, and finally implement them in program code. On the test execution side, we start by executing unit tests, followed by integration, system and acceptance tests.

The V Model (refer to Figure 11) is valuable because it highlights the existence of several *levels of testing* and delineates how each relates to a different development phase:

- Unit tests focus on the type of faults that occur when writing code.
- Integration tests focus on low-level design especially in checking of errors in interfaces between units and other integrations.
- System Tests check whether the system as a whole effectively implements the high level design, including the adequacy of performance in a production setting.
- Acceptance Tests are ordinarily performed by the business/users to confirm that the product meets the business requirements.
- At each development phase, different types of faults tend to occur, so different techniques are needed to find them.

Testing was never meant to be done after coding. Testing process also involves identifying what to test (test conditions) and how they will be tested (designing test cases), building the tests, executing them and finally, evaluating the results, checking completion criteria and reporting progress.
Not only does this process make better tests, but also many testers know from experience that when they start to think about how to test something, they find faults in the procedure for testing itself.

Moreover, if we leave test design until the last moment, we won't find the serious errors in architectural and business logic until the very end. By that time, it's not only inconvenient to fix these faults, but they have already been replicated throughout the system, so they're expensive and difficult to find and fix.
According to the V Model, as soon as some descriptions are available, we should identify test conditions and design test cases, which can apply to any or all levels of testing. When requirements are available, we need to identify high-level test conditions to test those requirements. When a high-level design is written, we must identify those test conditions that will look for design-level faults.

When test deliverables are written early on, they have more time to be reviewed or inspected, and they can also be used in reviewing the development deliverables. One of the powerful benefits of early test design is that the testers are able to challenge the specifications at a much earlier point in the project.

This means that testing doesn't just assess software quality, but, by early fault-detection, can help to build in quality. Proactive testers who anticipate problems can significantly reduce total elapsed test and project time. (Goldsmith's more formalized "Proactive Testing" approach)

However, there might be some aspects of development where it is not always possible to use the V Model. For example take Java, one of the most popular programming language. It needs unit, integration, system and acceptance testing. The V model in this case would not tell how to define what units and integrations are, in what sequence to build and test them, how to do the tests, etc.

After looking at the various models, in general we could conclude that a more descriptive, a more continuous model that would exactly describe a development process with its state would be needed. It should have the following properties:

- It should trace the entire development process continuously.
- It should allow for feedbacks.
It should exactly define a system with its attributes.

It should provide sufficient control to tune the system.

Should incorporate certain factors that are abstract (fatigue, pressure etc).

With the advent of system dynamics and its wide range of applicability, it was thought that system dynamics could well be applied to the modeling of software coding project so as to capture the dynamic behavior of its various elements.

2.7 System Dynamics Modeling of Software Projects

System Dynamic Models are based on the bottom-up approach to aggregate as many details of the project into the whole system dominated by internal interactions. They consider management problems at strategic levels and their main priority is to capture the more general aspects of project behavior that result from internal feedback processes and cause-effect relationships as shown in Figure 12.

![Figure 12. A Cause-Effect Relationship (Collofello, 2000)](image)

The figure provides an example of cause-effect relationship that might be observed during software development. It indicates that the time developers spend on quality activities impacts the number of undiscovered defects remaining in the software product. The “I” on arrow indicates that its an inverse relationship. We realize the
utilization of these models in complex models with feedback loops. Let us take another example shown in Figure 13

![Figure 13. Feedback Loops (Collofello, 2000)](image)

The loop with “+” sign is called reinforcing loop. As the developers perceive that their “ability to meet the schedule” is lessened, they spend less “time on quality activities” which results in increase in “undiscovered defects in product” which further lessens their “ability to meet the schedule” and so on. The second type of feedback loop called the balancing loop has the purpose of reigning in reinforcing loops. Balancing loops bring the system back into balance. The balancing loop is labeled with “-” sign. In this example, as the developers perceive that their “ability to meet schedule” is lessened, they spend less “time on quality activities” which they perceive to increase their “ability to meet the schedule.”

The first application of system dynamics to project management was proposed by Abdel-Hamid and Madnick (1984) leading to the development of a generic model of
software development process. The testing and practical application was based on the study of a real software project at NASA’s Goddard Space Flight Center. Further improvements led to a generic simulation model of software life cycle embedded within an expert system. The testing and validation of this was also done on a live project. They have done excellent work but did not consider breakdown of a project work. Also they assumed stable project requirements, which in case of most medium sized software projects is not the case.

Cooper (1993) has studied various system dynamic applications to software development projects. His focus is more on “work quality” and “time to discover rework” based on generic concept of work cycles. He found out that gains in project performance are governed mainly by directing efforts to increase work quality and detect errors earlier. He also suggested that rework generated in project remains undiscovered until later stages. He introduced the Project Management and Integration Model, which include design procurement, test, staffing categories and program management. The models, however, are not very detailed. So it is very unlikely that they are suitable to provide support at lower tactical levels.

As we look into these models, we find that the best way to use system dynamics is to take only one department, for example testing or staffing at one time and design it into intricate details so that we do not leave out even the smallest details and then based on that model decide as to what could be the right policy used to get maximum benefits. We would try to look and decide as to what should be the right staffing policy with respect to cost and time of a software project. The cost and time for the project are also calculated using Vensim (www.ventanasystems.com).
CHAPTER 3
PROBLEM STATEMENT

Recently, human factors in project management have gained importance as it is now recognized as the core of effective software project management (Oglesby and Urban, 1983). The reason being that personnel costs have started exceeding hardware costs. Chronic problems in software development and implementation are more frequently traced to personnel shortcomings. Information system staff sizes have mushroomed with little time for adequate selection and training. Software project managers find themselves paying more attention to human resource issues (Hannan, 1982). Hiring decisions involving timing and level are key to a company’s success. These decisions affect the quality of the product developed, the time in which it is developed and the total cost of the project.

We focus our attention on the dynamics of software project management using a system dynamics based approach so as to decide the exact hiring policy and the cost of that policy.

3.1 Brook’s Law

There is always a belief that by adding workforce, the time to complete the project is reduced. But that’s not always the case. As per Brooks’ Law, adding manpower at a later stage in software project can delay the project further (Pei, Hsu and Kung,
1999). Even though adding manpower increases the headcount, the newly appointed employees need to be trained, which consumes a vital experienced workforce. Naturally the overall productivity of the system is hampered. This has been observed to be true in a real software project, namely, NASA’s DE-A software development project (Abdel-Hamid, 1989). If there needs to be a policy suggestion as to how to deal with such situations, it becomes imperative to get a complete picture as to what exactly are the factors that might influence the development process. The policy then needs to be tested on a replica to know what the impact is going to be.

3.2 Integrative System Dynamics Model of Software Development

This model provides an integrative perspective into the development process. It integrates the multiple functions of a software development process, including both the management functions (for example staffing, controlling, and planning) as well as software production-type activities (for example, testing, coding, designing). Our focus will mainly be on the staffing and controlling part.

A detailed description of the model is given by Abdel-Hamid (1984). A model would be developed based on its description using and corresponding time and cost estimates of the development process would be made.

3.3 Factors Under Consideration

The model can be visualized as a collection of connected activities performed in a sequence, within the context of dynamic project environment. Every model developed in this area is different depending upon the project environment. The objective is to develop
a very generic model that describes the basic processes in a product development. Our aim in developing the model is to determine what would be the expected completion time and cost associated with a project depending upon the project environment variables which include the net hire rate, error detection rate, productivity, cost of the developers etc. One by one all factors in the model are examined.

3.3.1 Effort Required

All activities are assumed to require some finite amount of work expressed in man-hours. When a project goes into testing phase, errors are detected and this adds to the work to be done. The number of errors introduced could be a function of many factors such as fatigue, pressure etc. In our model, we have limited the scope to the dependency of errors detected on the number of lines of code developed.

3.3.2 Time in Days

The time required to complete an activity is a function of the staff/people allocated to that activity. Productivity can be measured in thousands of lines of code or in technical terms kilo lines of code (KLOC) per day. But for modeling purpose, we would consider it as a percentage of experienced staff member productivity. This means a productivity factor of 0.7 would mean that the person could write 70% of lines of code that a normal experienced person in a day. Using the productivity and available staff members, both of which vary continuously, we can integrate the development rate until the required man-hours are reached.

3.3.3 Staff Levels

The number of staff members available is determined by a two-step approach (Martin and Raffo, 2001). The number of experienced and new staff members is
maintained by integrating a set of rates that describe hiring, assimilation (training), and transfer and quit rates. We are not going to consider any transfer between projects as we are taking only one project at a time into consideration.

3.3.4 Productivity

The rate at which work is accomplished and the number of staff allocated determine the duration of an activity. To determine the rate at which work is accomplished, we use a factor called the productivity, which is defined in terms of thousands lines of code written per person per day. This would be one of the inputs. The value of this input is taken either from historical data specific to a company or people who are experts in determining those depending upon influencing factors for that input. Productivity is affected by several factors such as fatigue in a person, schedule pressure, learning, etc. In this research we will consider effect of schedule pressure on time, cost and quality of the software developed.

3.3.5 Testing and Error Detection

Any product developed has to go through testing phase during which errors would be detected and need to be corrected. This requires testers who have a totally different type of job than the developers. The developers pool and testers pool are assumed as pool of developers and both have the expertise to be shifted to either job, which is the case in Appian Corporation, Falls Church, VA (www.appiancorp.com). The error rate is expressed as a percentage of work done. Rework adds to original work and further delays the development of the product. The error rate can again be either taken from an historical data or any personnel in this field can give the Figures.
3.4 Problem Description and Objectives

The objective of this thesis is to provide the project manager with a tool that could help him/her to get a rough estimate for a bid. This tool can also be used to change parameters and see how to get project completed at scheduled date. It can also be used to monitor the current progress of the project. With these objectives in mind, we would try to develop a system dynamics model. The modeling as mentioned earlier would be done using Vensim. We will consider two cases:

*Case 1:* The hiring rate is constant and is not determined by the progress of the project. The productivity of the staff is also constant and is not a function of schedule pressure.

*Case 2:* Depending upon the state of the project and time remaining, the management has to take a decision if they need to hire more people or use overtime to get the work done.

As the management takes a decision, they also have to keep in mind that by asking people to work overtime, the work done by those developers might be influenced because of the pressure and fatigue creeping in due to late hours of continuous work. The productivity of the developers might get affected because of pressure. This in turn affects the performance of the system.

3.4.1 Performance Measures

*Time of project:*

Depending upon the hiring rate and policy adopted, the time taken to complete a project changes. This is measured in months.

*Cost of the project:*

This is a function of time and staff. Since the overhead cost is not dynamic, cost would change with time and number of people working on the project.
Productivity:

This is usually a function of three factors namely fatigue, ratio of new to experienced personnel and schedule pressure. The last two would be considered. A check has to be kept on the productivity so that productivity never falls below a certain level. If it does, people need to be laid off and in most cases it is the new hires that are fired because they are not contributing much to the organization but also need to be paid.

3.4.2 Assumptions

- All the staff members who are at the same level of experience have the same productivity. Testers and developers are not differentiated. The overall productivity is determined in terms of productivity of experienced personnel irrespective of whether he is in rework or development.

- No transfers between projects take place. Since we are modeling time and cost for a single project, we do not consider personnel being transferred from one project to another.

- When we employ new people, some experienced staff has to be dedicated in their training. Now while getting trained, we are losing overall productivity of the enterprise because the experienced employees are busy training the new hires. In Appian Corporation, on an average, 15 percent of the employees do spend an hour of their work time in solving problems or administering a technology class. To take that into consideration, we give certain percentage productivity to the new hires instead of decreasing the number of experienced personnel.

- The productivity is not affected by the decisions taken by management.

- Hiring rate is independent of quitting rate. In most models on project management, it is assumed that hiring rate = total quit rates of new and experience employees. In real world that is not the case. If an experienced employee quits, we do not necessarily want to get a new employee because he is going to take finite assimilation time and if the time remaining for project completion is less than the training time, it does not make sense to hire a new employee.
CHAPTER 4

SYSTEM DYNAMICS MODEL OF SOFTWARE DEVELOPMENT

4.1 Introduction

In chapter three we described the basic concepts of software development. In this chapter, we discuss the building blocks of the base model for the coding process in a software industry. The model discussed in this chapter does not consider any feedback loops. This model gives an approximate value for the time and cost associated with the project based on the current value of various parameters like hiring rate, cost of the developers etc. of the company. A project manager can use such a model as a reference when he tries to project a timeline and cost for a particular project. The feedback loops are discussed in details in the fifth chapter.

4.2 Model Building Blocks

The model discussed in this chapter consists of four sub-systems viz.

- Productivity Sub-System,
- Work Rate Sub-System,
- Workforce Sub-System, and
- Cost Sub-System.
The project definition in our model is the approximate number of lines of code. This value is set to 1000 Kilo Lines of Code or KLOC and is called ‘work to be done’. For coding purposes, we need to hire people if needed. This hiring process is described by the workforce sub-system is shown in Figure 14.

![Diagram of Workforce Sub-System]

**Figure 14. Workforce Sub-System**

Every organization starts with some experienced and new employees. These values are set using the variables *Initial New Developers* and *Initial Experienced Developers*. Over the period of time the organization hires new employees depending upon requirements and decisions taken by the management. The hiring rate is constant in this model. The value of this constant could either be the current hiring rate in the
company or some other value which the manager might think to be the rate of people getting hired in near future. The newly employed developers take certain training time, which is generally constant unless there is a change in the training methodology. In Appian Corporation (www.appiancorp.com), this is one month. We have also used one month in our model. During this training period, some of the experienced employees are engaged in the training of the newly hired developers. The productivity of the system as a whole decreases because these people are devoting their time to training which otherwise could have been used for development. Figure 15 shows how productivity is affected by the hiring of new employees.

We first calculate the fraction experienced in the organization and then use this value to get the productivity index from a lookup graph shown in Figure 16. On the x-axis we have the fraction experienced and on y-axis we get the productivity index. For example, when the fraction experienced is 1 (there are no new developers in the organization), the value of productivity index is 1. In the graph (refer to Figure 16), input means the fraction experienced and output means the productivity index. Other details of are explained in the equations describing the model. The variables with a “<” and “>” enclosing them are called shadow variables. These are the variables that connect one view with the other.

Whenever code is written, some errors escape undetected. The time to detect the first bug or error has a higher value because the product is in either in design and planning stage or the development has just started. But at later stages, which might be testing, the errors get detected fairly quickly. Therefore, the time to detect errors is not a constant but is expressed in terms of time of the project or the fraction of work done.
Initially, when the fraction of work done is 0, this time to detect error would be high. (For a mid size project, this is generally 5 to 6 months when the first modules are getting tested. Refer Figure 17).

Figure 15. Productivity Sub-System

Figure 16. Lookup Graph for Productivity (Ventana Systems, Inc.)
As the project nears completion, this time tends to zero because the work being done now is mostly testing and deployment. For this model the work quality is assumed to be 0.9. Figure 18 shows the work rate sub-system.
The cost of new developers and the experienced developers is assumed constant over the period of the simulation for this model. In chapter five and chapter six where the game mode of running the simulation is discussed, it is possible to change the value in the middle of the run. This is particularly significant because there is always a possibility that some people might get a raise during the project duration.

The cost sub-system is shown in the Figure 19. Here Cost of new developers and Cost of experienced developers are levels and increase at the rate equal to the developers in their corresponding pools multiplied by the cost factor for that pool. The total cost is just a summation of the two costs.

Both the project completion time and cost could have been plotted in one graph. But as we make more and more runs, such a graph would become crowded and unreadable. So they are plotted on different graphs.

Figure 19. Cost Sub-System
4.3 Model Equations

This section describes the equations used in the model. In the workforce subsystem there are some new developers and experienced developers at the start. New developers are trained for the training time and then they are moved to the pool of experienced developers. This is expressed, as a rate at which there is an increase in the number of experienced developers.

\[
Rate \ of \ getting \ trained = \frac{\text{Newly hired developer}}{\text{Training time}}
\]  

(4.1)

The hiring rate for new employees is constant and is measured in Developer/Month. As the new employees get trained, they should no longer be a part of the new developer pool. So for the level, Newly Hired Developers (Refer Figure 14), we need to subtract the rate of getting trained from the hiring rate. The summation of the levels, the newly hired and the experienced, is maintained in the Total Developers level.

We then calculate the fraction of employees experienced using the following equation

\[
\text{Fraction Experienced} = \frac{\text{Experienced Employees}}{\text{Total Employees}}
\]  

(4.2)

This fraction is used as the input for the lookup graph for productivity index. This graph takes the fraction experienced as the input and gives the value for the productivity corresponding to that fraction. This value is then multiplied with normal productivity to give the actual development rate in KLOC/Developer/Month (Refer Figure 15).

\[
\text{Development Rate per worker per month} = \text{Normal Development Rate Per worker Per month} \times \text{Productivity Index}
\]  

(4.3)

This is then multiplied by the total number of developers to get the current work rate of the entire organization.
\[ Work\ Rate = Development\ Rate\ per\ worker\ per\ month \times Total\ Developers \]  
\[ (4.4) \]

This work rate is now used as the workflow rate (rate at which work is done at a particular instant of time). Work is done at this rate and hence the work remaining level goes down. But rework is generated too. This adds to the work to be done. Hence for work remaining we have

\[ Work\ Remaining = Rework\ discovery\ rate - Work\ Flow\ (KLOC) \]  
\[ (4.5) \]

Understandably, exactly opposite would be the equation for work done.

\[ Work\ Accomplished = work\ flow - rework\ discovery\ rate\ (KLOC) \]  
\[ (4.6) \]

We need to know exactly when the project is done. This is accomplished by defining a variable called ‘project is done’. This is used as a Boolean variable and is set to 1 or zero depending upon the fraction of work remaining. In the model, we set it to be 1 when all work is done and 0 when even one line of code is remaining. This is achieved by IF THEN ELSE decision statement in Vensim.

\[ Project\ is\ done = IF\ THEN\ ELSE\ (Fraction\ Complete \geq 1,1,0) \]  
\[ (4.7) \]

Where,

The equation essentially means that if the fraction complete is more than or equal to one (project is done), set the value of ‘project is done’ to 1 else set it to zero.

And for workflow we have

\[ Work\ Flow=IF\ THEN\ ELSE\ (Project\ is\ Done, 0,Work\ Rate) \]  
\[ (4.8) \]

Thus, whenever there is a non- zero value for ‘Project is Done’, the ‘Work Rate’ would be zero. Otherwise the value of ‘Work Flow’ would be equal to ‘Work Rate’. 
4.4 Simulation Runs and Results

Having described all the equations, we now move ahead and try to test our model to different scenarios. Our aim is to determine what would be the best decision to complete a specified task at minimum cost or time with the project definition of 1000 KLOC.

Case 1: No new employees, 10 experienced employees and no hiring.

   Project Duration: 11.43 Months  Cost Associated: $ 400,313

Case 2: No new employees, 10 experienced employees and hiring @ 2 employees/month.

   Project Duration: 8.31 Months  Cost Associated: $ 505,594

Case 3: 10 new employees, 10 experienced employees and no hiring.

   Project Duration: 6.63 Months  Cost Associated: $ 448,795

Case 4: 10 new employees, 15 experienced employees and no hiring.

   Project Duration: 5 Months  Cost Associated: $ 419,078

Case 5: 10 new employees, 15 experienced and hiring @ 4 employees/month.

   Project Duration: 4.18 Months  Cost Associated: $ 420,984

Case 6: 10 new employees, 10 experienced employees and hiring@ 4 employees/month.

   Project Duration: 5 Months  Cost Associated: $ 479,128

From the results, it is very clear, if we do not have constraints on the number of experienced developers at hand; it is always recommended not to hire new employees (as shown by case 1 results). In the event of an earlier completion date and a limit on the number of experienced workforce available, we resort to hiring of new employees. As
indicated by the results for the six cases, if the scheduled completion date is not earlier than 9 months, there is no need to hire new employees. They do help to get the project done earlier, but cost more. Another factor to take into consideration is the training time. In this model we have taken it as 1 month. But this might change and this would directly impact the projected completion date. The graphs for work accomplished, total cost and work flow for the six scenarios that were discussed, are given in Figures 20 through 22.

**Figure 20. Work Accomplished**

Runs 1 through 6 corresponding to cases 1 through 6 are plotted in Figure 20. The line with the least gradient is run 1 and it increases to run 6. If we look at Figure 20, we see the graphs are linear and they become constant after reaching the project definition of 1000 KLOC. The reason is that in this model there are no feedback loops and the equation for development rate is linear and hence the work-accomplished graph is also a straight line.
If we look at the graph for the total cost, we see that it continues even after the project is done for that particular run. The reason is that Vensim plots the behavior of variables. In the context of above graph it means that if the project definition was changed from 1000 KLOC to any other value, if other conditions for the run do not change, then the time taken would be somewhere along the same plot for that run. So for example, for run 1 that corresponds to case 1, if the project definition was 2000 KLOC, the time required could follow the same line as plotted for run 1 in the above graph.

Figure 22 gives the graph for the work flow in (KLOC/Month) plotted on Y axis, which is the rate at which work is done. It keeps on increasing for runs 2 through run 6 (which correspond to case 2 through case 6), which implies that either new developers
are getting hired or the new developers are moving to the pool of experienced developers resulting in the increase in the productivity of the system.

Figure 22. Work Flow

Naturally, for run 1 or case 1, this is a straight line because we have the hiring rate as zero for that run and there are no new developers at the start of the run. So this line is horizontal. For run 4 (case 4) we have the hiring rate as zero but the work flow graph for that run is not a horizontal line because in that run we started with some new developers which after a month moved to the pool of experienced developers there is a change in the overall productivity of the system. If we observe closely, the plot for that run is steeper at the start (indicating the moving of new developers to the experienced developers pool), and then it becomes a straight line because there is no more change in the productivity of the system. The point at which the work flow lines in the graph drop to zero is the time at which the value for project is done variable becomes zero and we set the value of work flow to 0. The spikes after completion time indicate small rework that may not have been done when we concluded the project. Typically these are the bugs the client points out.
after the product is sold and the client starts using those.

This chapter dealt with our first model. In Chapter Five and Chapter Six we would incorporate feedback loops into our model and provide explain the different modes in which the model can be run depending upon the use of the model. We would also explain the need and development of a graphical user interface to the model.

4.5 Verification of Results

The model was verified by comparing the simulated values with the expected values. For example, in case 1, since no new employees are hired, there are 10 experienced employees paid at $3,500 per month.

\[ 11.43 \times 3500 \times 10 = $400,050 \]

The time taken is equal to

\[ 1000 / (0.9\times10) = 11.1 \text{ months.} \]

The actual result gives the value as 11.43. If we look at figure 22, we can see that the this value corresponds to the second spike for work flow.
CHAPTER 5

MODEL WITH FEEDBACK LOOPS

5.1 Introduction

In this chapter, we try explain how feedback loops work in the software development process and incorporate them in our basic model presented in Chapter four. The results of the feedback loop based model will be compared with the earlier developed model. The models have been enhanced by providing a graphical user interface. The model can also be run in its entirety by changing various parameters (Synthesim mode). It may also be useful to make parameter changes in a model from a certain point onwards and see its effect on the project performance (Game mode). Thus, overall there are three models:

- Bidding model with user interface.
- Model with feedback loop in *synthesim mode*.
- Model with feedback loop in *game mode*.

5.2 Feedback Loop in Software Project Development

The model discussed in chapter four did not have any outputs affecting the input parameters. In real life, software development project, this is not the case. The input parameters are continuously monitored and their values are changed depending upon the
state of the system. But it is the project manager who generally changes the values of the input parameters, based on his personal experience in that field (as is the case in Appian Corporation, VA). Instead of relying on an individual’s view, we have tried to give a systematic representation of the feedback that takes place in a software development process. Refer to Figure 23.

![Figure 23. Feedback Loop](image)

As we can see, we first provide the project deadline indicated by the variable \( P_{\text{deadline}} \). The units of this are in months in our model. This is initially set to the value decided by the project manager. Now with the current time and project deadline we calculate the time remaining. This is calculated using the following equation

\[
\text{Time Remaining} = \text{Project Deadline} (P_{\text{deadline}}) – \text{Current Time.}
\]  

\( (5.1) \)
Now if the project deadline is overrun, this value could be negative. Hence instead of using this equation, we modify it as

\[ \text{Time Remaining} = \max(0, \text{Project Deadline} - \text{Current Time}) \]  

(5.2)

This equation implies that once the project deadline is reached, we no longer keep calculating the time remaining but make it 0. We already know the work remaining which is a level whose rate is

\[ \text{Work Remaining} = \text{Work Remaining} + \text{Rework Discovery Rate} - \text{work flow} \]  

(5.3)

The work rate therefore required to get the work done on time is

\[ \text{Work Rate Required (WRx)} = \frac{\text{Work Remaining}}{\text{Time Remaining}} \]  

(5.4)

But when the scheduled time is reached, the time remaining would become zero. (Refer equation 5.1). So we modify this equation as

\[ \text{Work Rate Required (WRx)} = \text{XIDZ (Work Remaining, Time Remaining, Max Work Flow)} \]  

(5.5)

XIDZ is a Vensim function that means X If Divided by Zero. When applied in above manner it means if Time remaining is Zero, use Maximum Work Flow. But if the project is done earlier than the deadline, we need to set Work Rate Required to zero. So the equations is again modified

\[ \text{Work Rate Required (WRx)} = \text{IF THEN ELSE (project is done, 0, XIDZ (Work remaining, Time Remaining, Maximum Work Flow)} \]  

(5.6)

Knowing the work rate required, productivity and also the maximum workforce that a company can have, we now calculate the workforce required to get this work done.

\[ \text{WFReqx} = \min(\text{Maximum Developers, Work Rate Required/Normal} \]
To calculate the indicated workforce, we take into consideration a human intelligence factor called the Willingness to Change Workforce. This has two components in itself.

- Willingness to change workforce I
- Willingness to change workforce II.

These are industry dependent factors and would vary from organization to organization.

Willingness to Change Workforce I compares the time remaining to the total time required to get a new software developer to work efficiently (sum of hiring time and training time). This ratio then determines the value of this factor.

\[
\text{Time Ratio} = \frac{\text{Time Remaining}}{\text{Hiring time} + \text{Training Time}}
\]

\[
\text{Willingness to Change Workforce I (WCWI)} = \text{WCWI\text{Lookup} (time ratio)}
\]

WCWI\text{Lookup} is a lookup variable, which plots the values of WCWI for different values of time ratio.

The second factor takes into consideration the fraction of the project complete.

\[
\text{Willingness to Change Workforce II (WCWII)} = \text{WCWII\text{Lookup} (fraction Complete)}
\]

Now the actual value is

\[
\text{Willingness to Change Workforce} = \text{MAX} (\text{WCWI}, \text{WCWII})
\]

The indicated workforce is calculated as

\[
\text{Indicated Workforce} = \text{WCWF}\times \text{Wfreq} + (1-\text{WCWF})\times \text{Total Developers}
\]

But this should only be the case when total developers are less than required workforce, else it should be equal to workforce required to meet the scheduled time.
The equation therefore gets modified as

\[
\text{Indicated Workforce} = \text{IF THEN ELSE (Total Developers}< WFreqx, \\
\text{CWFX*WFreqx+}(1-\text{WCWFX})* \text{Total Workforce, Required Workforce)}
\] (5.13)

This indicated workforce is now used to calculate hiring rate. For this, we need to first check if the indicated workforce is greater than the current workforce, only then we hire otherwise we need to lay off people because we are using more resources than required (which is the indicated workforce). The hiring rate therefore, which was constant earlier, now is

\[
\text{Hiring Rate} = (\text{Indicated Workforce} – \text{Total Current Workforce})/ \text{Hiring Time})
\] (5.14)

If the total workforce is greater than indicated workforce, then we should not hire. So the equation gets modified as

\[
\text{Hiring Rate} = \text{IF THEN ELSE (indicated workforce}> \text{total workforce, (indicated workforce – total current workforce/hiring time,0)}
\] (5.15)

We also put a cap on the hiring rate because even if the model suggests a hiring rate of say 100-developers/ month, it might not be possible. So

\[
\text{Hiring Rate Actual} = \text{Min} (\text{Hiring rate, Max Hiring Rate})
\] (5.16)

To incorporate pressure effect, we first set the maximum pressure level to say 5 (this can be changed). The actual pressure is calculated using the equation provided by Vensim model of software development

\[
\text{Pressure Exerted} = \text{ZIDZ (Work Rate Required, Work Rate with Indicated Workforce)}
\] (5.17)
ZIDZ means zero if divided by zero.

Work Rate Required is calculated earlier in equation 5.4. Work rate with indicated workforce is simply the indicated workforce (equation 5.11) multiplied by productivity. The error rate also is no longer constant. It is now a function of number of lines already coded, which is very logical because as the number of lines of code increases, the chances are more error lines of code are introduced.

The schedule pressure is taken as a ratio of required workflow and normal workflow (Ventana Systems). Thus,

\[
\text{Schedule Pressure (Pressure index)} = \frac{\text{Required Workflow}}{\text{Normal Workflow}}
\]  

(5.18)

Increased pressure would result in overtime,

\[
\text{Overtime} = \text{Overtime Lookup (Schedule Pressure)}
\]  

(5.19)

The lookup graph is shown in Figure 24.

![Graph Lookup - overtime lookup](image)

**Figure 24. Lookup Graph for Pressure (Ventana Systems)**

On the x-axis, there is the ratio of required workflow and normal workflow (or pressure
Productivity now becomes

\[ \text{Productivity} = \text{Normal Productivity} \times \text{Overtime} \quad (5.20) \]

This value is then used in the model instead of a constant value, as was the case in our first model.

To handle excess developers, we use the following equation

\[ \text{Dismissal Rate} = \frac{(\text{Total WorkForce} - \text{Indicated Workforce})}{\text{Dismissal Time}} \quad (5.21) \]

5.3 Feedback Model with Synthesim Feature

The feedback loop is incorporated in the model and a graphical user interface is provided to interact with the model. Figure 26 shows a view of this model. The model is run in synthesim mode (model simulates for the entire length of the simulation when the value of variable is changed using the sliders).

The sliders at the side can be moved to simulate the model instantaneously and the graphs change as we change the value of the variables. The graph named cost shows the increase in cost every month while the graph named total cost shows the summation of costs till a particular time. There is another graph called cumulative cost, which is plotted considering that all the developers come from the same pool (as proposed by Vensim). This is plotted just to compare it to the cost of the project given by our model. All the costs are in Millions of dollars. There is a graph indicating project is done. This is a variable and its value becomes 1 when project gets done. We can see that the project gets done somewhere between 10 and 11 months. To see the actual value of it, we try to
get the dataset representation of the variable as shown in Figure 25.

Figure 25. Tabular Data of Project is Done Variable

Figure 26. Model Run in Synthesim Mode
If we look at the graph of dismissals, it shows a spike at about the same time the project is done. This is because when project is done, the work to be done becomes zero. Naturally the workforce required (indicated workforce) becomes zero and there is an excess of developers in the pool. At that instant, the total developers become greater than the developers required. Hence the dismissal rate takes a high positive value and we see the spike. In a real scenario, these people get transferred to a different project. There are sliders for project definition, productivity (starting value, completion date etc. We can see that the starting value for project definition is 1000-kilo lines of code. We can move the slider and Vensim would simulate the model to plot the graph accordingly. Thus in this model to present 1000 kilo lines of code and given condition, it take about 11 months and the cost incurred is around 4.8 million dollars.

5.4 Feedback Model in Game Mode

Vensim model in game mode is used as a control tool. The reason is that, in game mode we can change the value of certain factors and let the model run from that moment of time instead of simulating the model for the entire length of the project with those values. Such variables are called game variables. In this model, some variables like cost of developers, project duration, etc. are made game variables.

Let us take an example to explain our model in game mode. The time step is set at 0.5 months. This means that every 0.5 months, this model will pause and if we want to change any values of key variables, we can do that and let the model run from there onwards. At the start of the simulation we have the deadline set to 10 months. Let us say
that after 2 months the client has some financial crises and he is going to pay the promised amount over a period of 14 months instead of 10 months. In that case, it does not make sense to keep hiring at the same rate and completing the software in 10 months and then if there is no other project, laying people off. Instead we now need to adjust our parameters to accommodate this change. Once the model is developed, this is taken care by the model. In Figure 27, we can see the deadline is set to 10 months. We run the model step by step till 2 months and then when the simulation stops at 2, we change the project deadline to 14 months and keep running the model again. Now we can clearly see how the model behavior deviates from the earlier model behavior (refer to Figure 28). The hiring rate drops down as compared to the earlier simulation and also the cost goes down. All we did was changed the project deadline and then the model gave us the expected behavior as well as the parameter values that are changed or need to be changed. One can clearly see the utility of such models. Once developed, there is no need for us to sit down and calculate values. Such models can definitely be used for reference if not used as the sole source of making decisions.

Figure 27. Feedback Model in Game Mode with Ten-Month Deadline
5.5 Results

If we look at the first model we developed in which we had no feedback loops, we had a starting hiring rate that never changed over the length of the simulation and hence we could only project the deadline of the project. Such a model is used when we want to go for a bid and need to quote certain man-hours that would be required to complete the project. This is exactly how it works in a consulting firm like Appian Corporation, wherein the company quotes the number of consultants it would need to complete a project and the number of man-hours each one would put in over the period of the project. The man-hours are then used to calculate the cost.

Once the project is awarded, the firm needs to deliver or try its best to deliver the project at the given time. For this it might need to hire new people or make current workforce work overtime. Now if we look at the second model, the model is an ideal case
of development. We call it an ideal model because no matter what deadline is set, it will always give us the value of parameters and end the project at the exact deadline. But this is assuming the fact that the existent parameters can be stretched to the calculated values for exact completion.

For example, if we look at Figure 27, we see that the graph for hires shows that at the start of the project, the hiring rate should be around 50 developers per month for the project to be completed in 10 months. Now this is a suggested value and it may not be possible to hire so many people. In that case, the project gets delayed. Now if this deadline shifts to eight months, the hiring rate even goes up (refer Figure 29). The model will always suggest the value that is necessary for project completion on time. The actual hiring however determines the length of the project.

![Figure 29. Hiring Rate with Eight-Month Deadline](image)
A closer look at the graph for the hiring rate (referred hires in Figure 30) indicates that it keeps on increasing as the project deadline decreases. But there is a limit to which this can be increased and that keeps a cap on the date at which the project can be completed. This limit is placed by the variable called maximum workforce variable. Thus we see that this model can suggest the optimum value for various variables for proper project completion but it might not always be feasible. In a nutshell, the second model provides a tool that aids the project manager take decisions, the final decision though rests with the project manager.

This decision-making ability is incorporated in the third model. Here we set the time step to any value. The idea behind it is that, the model can now be run for specific time interval and now we can change the value of decision variable and then let the model run ahead. Take the example of salary changes that might take place through the course of the project, for example, due to slump in economy, the company might decide a salary percentage cut for some time. To incorporate this, we set the time step to one month and then every one-month this model would run and then wait. We can then change the values for our variable and again allow the model to run one more step. This is clear from Figures 30. Thus we can see the gaming feature can be used to take decisions and see their impact on the project. In Figure 30, the project has reached the fifth month of simulation and the model waits for user input, either to change any variables or just click on the arrow next to the red stop button to move another step ahead. Now we change the value for cost of new people to say $2300 from $2500 per month and allow the model to run to the end of the simulation (when value of project is done becomes zero).
Figure 30. Project State at the End of Five Months

We can see how with the aid of system dynamics tool that it becomes possible to model the software development process. This is a powerful technique and can aid a project manager to a great extent in making a correct decision. This dealt with the feedback loop and various modes of use of the model. Chapter Six gives a detailed description of the various graphical user components used in our models and the importance of making an effort to provide such a user interface.
6.1 Introduction

There are three modes in which a project manager would use a system dynamics model proposed in this thesis.

- To get a rough estimate of the project duration and cost for bidding.
- To expose the model to different conditions and see how the system would behave to those changes.
- To monitor the ongoing project.

The models developed have eleven views and fifty-eight variables. If such a model is given to the project manager and he/she is required to search for variables to change their values, it would take a great deal of his/her valuable time. It is very likely that he might never use it. A manager would naturally prefer a user-friendly interface where he would be able to plug in the values and get the results in terms of graphs or comparisons. For example, if the manager wants to see the impact of a hiring freeze on the project time and cost, he should not be required to go into the model source, look for the variable hiring rate and then change it. Rather, he would prefer if he could change the hiring rate just by moving a slider and see the changes in the cost and time for the project. Keeping in mind the actual use of the model and the people who are likely to use it, we have built
graphical user interfaces for our models. The *Synthesim* feature of Vensim is used for that. When we use the Synthesim feature, for changing the value of a variable, we are not required to stop the simulation and run it again. Instead, the software automatically runs the simulation for the entire length of time as soon as we change the value of the variable by means of a slider. This is a very unique and useful feature of Vensim. In the model used to monitor the development process, we have made use of the *game feature* in Vensim. When the model is run in game mode, it runs for a specified time interval and then waits for the user input. The value of variables can then be changed and we move another step ahead. The difference of this from Synthesim feature is that, in here, it does not run the model again for the entire length of the simulation. Vensim just changes the value of the variables at the instant they are changed and uses those values for the remaining time. For example, after certain amount of time, due to management decision to lay off developers, we might want to stop the model at that particular time and change the dismissal rate to a positive value. This is possible using the game feature of Vensim. Thus we can continuously monitor the progress of the project.

### 6.2 Interface Development Tools

#### 6.2.1 Views

As the complexity of the model increases, the model might get very crowded and difficult to look at. Also we find it difficult to actually locate a place where we did not connect the arcs properly because there is already a mesh of arcs. A more refined approach supported by Vensim is separating logically related blocks into a single view. A view can be defined as a single viewable workspace. As a standard practice we generally
group logically related blocks into a single view and connect them to other views by using shadow variables that are explained in the first chapter.

**6.2.2 Custom Graphs**

Custom graphs are used to customize the content of the graph to show the selected variables, runs, and style, in one graph. In Vensim, if we select any workbench variable and click the graph tool on the left toolbar, it shows the graph only for that variable. But sometimes we might want to look at two or more variable graphs at the same time to track as to if there exists any relationship between those variables and how one reacts to the change in other. For example, we might be interested in tracking both the Indicated Workforce Level (workforce required to complete the project on time) and the actual workforce level to see how much is the difference. For that we use the Graph Control in Control Panel and there we add a new custom graph. We select the variables of interest to be displayed as shown in the Figure 31. Here we have tried to track one more variable that is the difference in workforce. Then when we run the model we get the graph as shown in Figure 32.

![Figure 31. Custom Graphs](image)

65
6.2.3 Sketch Comments

The various blocks of Vensim may not mean anything to an end user who does not have a working knowledge of Vensim. For example, if we present this model to a project manager we might want to provide instructions along with it so that it guides him how to use the model. To achieve all this, we use the Sketch Comments tool of Vensim. This is an example how we might want to have the welcome page (refer to Figure 33). There are two modes of using the Sketch Comment tool. Either it can be used simply to display some information or link to some other view in the same model. For example, the information in simple text just provides the information while the Control Panel box is
linked to the Interface view of the model. When a user clicks on the “Control Panel”, he is taken to the User Interface view of the model.

Figure 33. Welcome Screen

Figure 34. Control Panel
When we click on “Click here for instruction” as shown in Figure 34, we would see a Figure as shown in Figure 35. Thus we can see how we can navigate different views using the comments tool besides providing information to the user.

![Instructions to use the model](image)

1. First go to Windows Menu and select "CONTROL PANEL."

2. Unload any previous runs by selecting the Datasets tab from Control Panel and selecting all the runs in the right-hand side text area and transferring them to the left using the "<<" arrow key.

3. Select the "Run Synthesim" submenu from the main "Model" menu.

4. Click here to go to the "Interface". This view is provided with sliders to change values of variables and associated graphs are plotted on change. To get tabular data for those graphs, click on the appropriate link.

**Figure 35. Instructions View**

6.2.4 Input-Output

We can customize the way our sketches work with simulation models by adding input-output controls. These can be alongside the model or in separate view. Using these tools we have tried to build a control room for our model for modeling simulation inputs and viewing simulation results. These controls are not a part of the model structure and so do not influence the model behavior. These controls also easily adapt to changes in model structure. If the name of a variable is changed, the corresponding control will
automatically be updated. If we look at Figure 29, the sliders are the input controls while the graphs shown are the output controls. Input/Output controls are the best mechanism that could be provided to someone who does not need to know any intricate details as to how the model works but still use it effectively.

6.2.5 Gaming Control

Sometimes it is quite possible that instead of allowing the simulation to run till the end, to use the simulation model as a monitoring tool, we want to stop the simulation after a particular time interval and then change the value of certain variables. For example, say in a 12-month project, after 3 months, the company decides not to hire anyone anymore. It means that the hiring rate value should drop to zero irrespective of anything else. For that what we do is, we change the type of the variable to a Gaming Variable. We set the Time Step (time interval for which the model will run and then again wait for user input) to say 1 month. So when we run the model, the model runs for 1 month time period and then waits for the user to either change the values or just run for another month. This is precisely what needs to be done because deciding some parameters beforehand and those remaining the same for constant for the remaining duration are never the case. If we look at Figure 36, we can clearly see that after 0.0625 months the simulation has stopped. Here advertising spending is set as the gaming variable (which is also the input control here). Just by means of a slider we can now set the value of advertising spending to some new value.
With a large set of useful tools available, user-friendly models were built, which, when opened by an end user, display a set of interactive tools. We have explained the working of these models and their characteristics in Chapter Four and Chapter Five. This chapter gives us a better idea as to what the different graphical tools in the models mean and why an effort was made to build interfaces to those models.
CHAPTER 7
CONCLUSIONS AND FURTHER RESEARCH

7.1 Conclusions

The basis of this modeling is the conceptual software development model put forth by Abdel-Hamid (1984). This model is then modified as per suggestions made by researchers such as Martin and Raffo (2001), who have been studying the system dynamics application to software project management. The application of system dynamics to software project management is relatively new and people have been very apprehensive in using the results provided by such models for actual decision-making. The reason for this is that certain factors like fatigue, due date pressure, development rate that have been used in the modeling are not the same for different developers on a project. However when we model the project, we are assuming that the level of performance of all the developers in a pool is the same and that they feel the same effect of the above mentioned factors. However, this model can definitely be used to expose the system to certain scenarios and see how it affects the project, which may not be feasible otherwise. This approach provides him a vehicle to see how the changing hiring rate would affect the system behavior (project time and cost), which would otherwise be not feasible. The model also provides a rough estimate of completion time and the associated cost for a project given the coding requirements. This could be helpful when a manager
wants to bid for a project. Our contribution here has been developing a comprehensive model for project management that tries to capture most of the factors that affect the development process. We have also taken into considerations the recommendations given by authors over the years and accommodated those in the model suggested by Abdel-Hamid (1984).

Similar attempts to modeling have been made typically by people who invented the software tools for system dynamics like Vensim, Stella, and Powersim etc. The model proposed by Vensim Inc is also based on the concept put forth by Abdel-Hamid (1984). However, they did not incorporate suggestions put forth by researchers in the field of system dynamic based modeling. There were a couple of basic problems with that model that we tried to eliminate in our model.

- The model proposed by Vensim does not separate the pool of experienced and new developers and use total developers to calculate the cost of project. This does not reflect the real life situation. We have different pools for new and experienced developers and we take into account their differing hourly cost.

- The model proposed by Vensim uses normal productivity factor of 1 (or some constant value) to calculate the indicated work force (This is the ideal workforce that we might want to complete the project at the right time irrespective of the cost). The productivity should be calculated at real time depending upon system parameters like ratio of new to experienced developers, fatigue factor, overtime factor and pressure etc.

- The willingness to change workforce factor in their model is not a component of two different factors as we have taken. Their model does not have the capability to make decisions as to willingness to change workforce depends upon the ratio of time remaining and sum total of training and hiring time. This is critical because not only do we want to take decisions depending on what percentage of the project is remaining (which determines value of willingness to change workforce I) but also if hiring new people and getting them trained would not take more time than the time remaining (willingness to change workforce II). In the model proposed by Vensim, they just consider the factor willingness to change workforce I.
The model by Vensim uses a constant value for error introduced. In our model we have changed that and it is now a function of the number of lines of code developed and is closer to real life project situation.

7.2 Further Research

If we try to model every possible factor that affects the system, the model would become extremely complex. Taking into consideration the time constraint and the scope of our research, certain assumptions were made when we developed the system dynamics based model. This is just the base model and one can always build on it. We would try to give a list of areas where there is a scope of improvement of this model.

- One area worth studying is the learning curve and how it affects the development rate over the training period. Here in this model we have assumed that over the training period, a particular productivity applies to a new developer. Instead this could be made a function of the learning curve.

- We have separated the developers into two pools, experienced developers and new developers. There might be a need to create an intermediate pool of developers, whose productivity is greater than a new developer because he has undergone training but less than an experienced developer because experienced developer may never have some issues as a programmer who is just out of a training academy in an organization. A more refined approach would be to base the productivity of a developer on the longevity of the developer in the organization.

- The fixed component of cost is not taken into account. The fixed cost component is generally a linear function of time. For example, consider a company rents a place to accommodate new people on the project. This amount the company pays is a linear function of time.

- The only variable cost component that we considered is the developer’s cost. To give an exhaustive model, quite a few variable costs should be taken into account.

- For example one could consider the variable cost of company paying for the flight the consultant takes every week to the client site.

- The model that we have given does not consider the communication overhead in an organization, which is also a determining factor in the project duration. One could always look into incorporating that factor in the model. In every project, there are meetings held to discuss the planning and in most cases there are weekly
updates and reviews with the client. These are the communication overheads that could be taken into consideration

- Our model does not consider multi-project situations and therefore does not assume that the developers are transferred between projects. An improvement over our model could be to consider two projects being done concurrently and then look into the transfer of developers between projects into account.
REFERENCES


Appian Corporation, http://www.appiancorp.com


76
