The Wizard of Oz & System Dynamics'

Application to System Test*

M. Stephen Kendall
Fusion-io
1000 S. McCaslin Boulevard
Suite 301
Superior, CO 80027 USA
720-458-4490
skendall@fusionio.com

Abstract

Introducing System Dynamics(SD) to solve a complex problem is difficult in two ways: 1) modeling the problem behavior, and 2) selling this approach as a desirable alternative to past troubleshooting methods. Adding the new concept of SD to the problem-solving mix often results in resistance. The perception is that other solutions worked in the past and there isn't time to learn new methods. This classic Limited Growth Archetype is best managed by addressing the balancing loop factors that limit adoption of change. In other words, the change agent needs to identify opponents and change their minds. In this paper I suggest that there is another way: provide the concepts and lessons without recipients being aware.

SD was introduced to a system test organization at a large corporation. The test organization was continually blamed for products being late to market because product validation took too long. The director of system test asked his managers to look at how to reduce the validation time. I raised the question, “What makes us think that anything we do will reduce the validation time?” Thus began the trip down the road of introducing SD as a means of understanding the product validation process.

During this journey, which lasted four years, successes and limitations were encountered. The final result was not at all what was envisioned when the effort began.

The final solution was to implement System Dynamic modeling behind the scenes so that test management and test leads used the results and had discussions with Engineering and Program Management without them being aware that those discussions were based on the SD problem-solving concept.

This paper describes the approach that was used to introduce and apply SD tools to the system test process.

Introduction

The Wonderful Wizard of Oz by L. Frank Baum, tells of how Dorothy and her dog Toto meet the Tin-man, the Scarecrow, and the Cowardly Lion, each of whom, like Dorothy, has a difficult problem to solve. They hear that the Wizard of Oz, who lives in Emerald City, is who they need

*The author conducted this work while in the employ of Oracle America and Sun Microsystems.

to see for help. After a difficult journey they arrive at the palace in the Emerald City and eventually stand before the Great Wizard as he blusters and bellows until they inadvertently learn that he is nothing more than a kindly old salesman hidden behind a green curtain. The power and wisdom they sought was not from a mighty Wizard but from a little old gentleman!

A way to apply SD in the workplace is similar to what was done in the Wizard of Oz: portray SD ideas as coming from knowledgeable members of the community, while hiding the power and insight gained from the models and SD disciplines behind the green curtain.

One of the biggest challenges for System Dynamics is to present the results in a manner easy to understand and not alienating to the intended audience. Furthermore, the results of any System Dynamics effort must be applicable. If not, the exercise, no matter how revealing, will be disregarded and the decision makers will retire to their comfortable decision-making tools based upon their current mental models, no matter how far out of touch they may be with what needs to be done.

As a member of a system test organization I introduced and applied SD tools and approaches to address the reduction of the duration of product validation. I investigated available generic models and used them as a reference point as I led the test engineers, testers, and management through documenting the process and creating their own process model so they would have a sense of ownership.

Product validation is a test process that begins when a product leaves development engineering and formally enters a system test phase where functional and fault testing are conducted on the product to be released. After product validation demonstrates a defined level of product maturity, the product is released to the field.

The Program Teams had the perception that the test process itself was prolonging the validation effort. Heroic attempts to lessen the validation time reduced the number of test cases, began test earlier, and used new development processes. But all fell short of the intended goal of reducing product validation time. The test managers were challenged to find ways to improve their processes in order to reduce the duration of test.

The following question challenged the conventional wisdom: “What makes us think that anything we do can reduce the validation period?” With this question System Dynamics entered the discussion.

Few problems seem more complex than managing a system test environment. Most vexing is that, while test managers have a firm idea of what to do to promote and measure test effectiveness, no matter what they implement, test continues to be viewed as causing delay in the release of new products. What causes this phenomenon? Why don't test process improvements fix this problem?

The system test group was new to System Dynamics' concepts, tools and vocabulary. So I provided a general overview and then used Causal Loop Diagrams and Stock and Flow models to help the organization understand and quantify the major influences underlying the duration of product validation time.

This paper describes the approach we used in applying System Dynamics within the system test environment and the changes that came about because of these efforts.

Introduction to System Dynamics

System test subject matter experts convened: managers, test leads, testers, and test architects. I introduced the System Dynamics vocabulary and described how Causal Loop Diagram (CLD) and Stock and Flow modeling tools work. The objective of the SD familiarization was to make the participants familiar with SD terms and general concepts. In retrospect this process could have been simplified by leaving out discussions on archetypes, and the origins and theory of SD.

The next step was to build an open form CLD of the entire Product Validation Process. Some SD modelers prefer to begin with stock-and-flow models but the limited SD knowledge of the participants and the free-flow manner in which the discussion began suggested use of the CLD as a starting point. The process began with the question, "What impacts how long it takes you to complete all your tests and how do you know when a product is ready to ship to the field?" All of the subject matter experts' responses were consolidated on a single board. Definitions of the factors were discussed to ensure everyone was talking about the same thing. Next the factors were connected to identify inter-relationships and whether these relationships provided positive or negative reinforcement (O=negative or S=positive).

The Causal Loop Diagram (CLD) (Figure 1) enabled management and testers to better understand the complexity of their process. When he first saw the CLD, one manager commented , "Where do you start?", which was exactly the point. How can a process this complex be managed by adjusting just one factor and expecting to have a predictable outcome?

The factors exogenous to the test process were color coded red. The colors indicated that over half of the factors influencing the duration of product validation were outside the control of the test organization. No wonder progress was so hard to achieve!

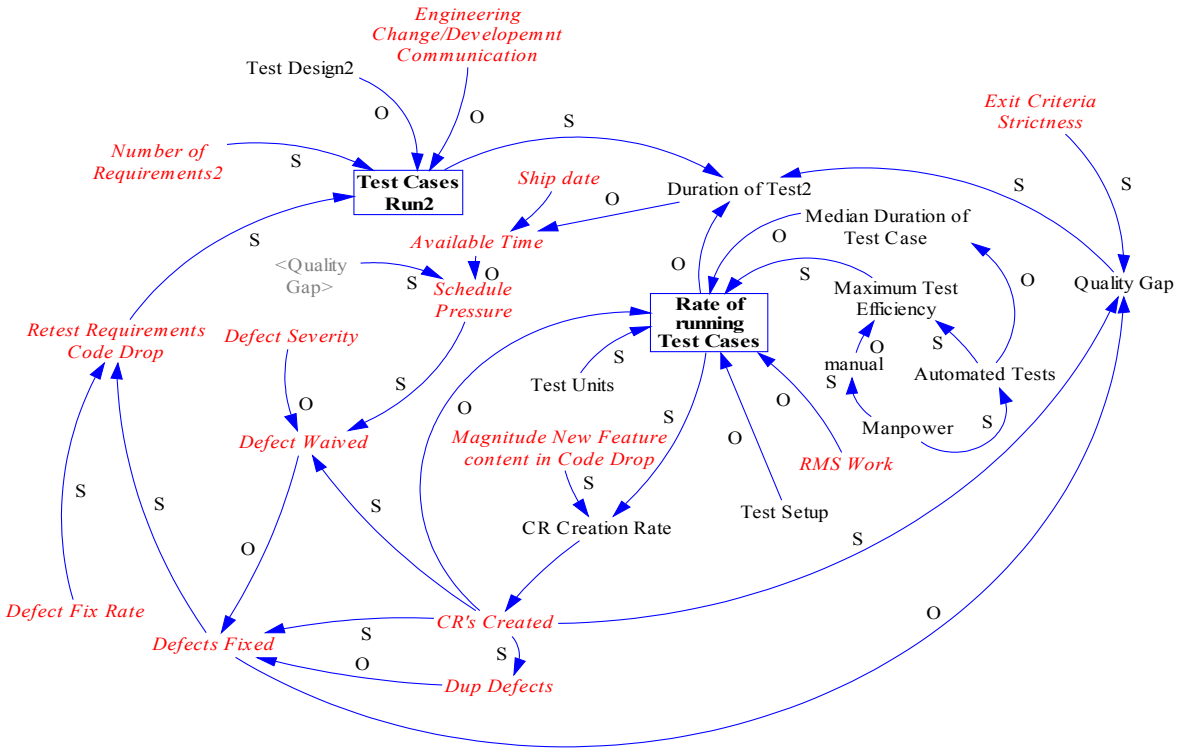

Figure 1

Building a PVM Model

Based upon the factors identified in the CLD, a stock-and-flow model was then built to quantify the factors and assess how sensitive the model was to each of them. Individual test departments were interviewed independently to quantify their values. Interestingly enough there wasn't any disagreement about the influencing factors. The test groups only differed on the value they assigned to each of the factors, which was driven by the product families they tested and the complexity of the system environment in which they operated. The resulting traditional program management model reflected feedback from test to development engineering and back to test. The model also included the product validation exit criteria since the program team can change the duration of the validation time by redefining what is good enough to ship to the field.

Based upon historical data, the model indicated that the duration of product validation depended primarily upon the number of defects found during test and the rate at which they were fixed. Test only became the bottleneck when a product entered test at a high level of maturity, which was influenced by the complexity of the design, the experience level of the developers and how development viewed test. If test was viewed as a customer then more due diligence was exercised during development to ensure the product was ready for test, resulting in a more mature product. When test was the limiting factor the duration time was driven by the number of test cases to be run and the rate at which the tests could be completed.

An engineering architect was intrigued by the discussion of the impact of the feedback loop between test and engineering. He knew of the feedback loop but was interested to learn how it behaved and how it affected the process validation time. He asked to use the model on a product currently in test to see if test was indeed not responsible for the length of the validation time.

The product's test parameters were fed into the model and the model's output was compared to the program schedule. The results indicated that the program was behind schedule mainly due to the low rate at which discovered defects were being fixed. This process delay caused test not to be able to complete its assigned tasks in a timely fashion. Engineering resources were shifted from another program to address the queue of defects and the program of interest came back on schedule.

During the same period a second program was examined. Its limiting factor was identified to be the volume of defects being discovered in test. While this program was too far into the program schedule to take corrective action, the program team was able to adjust the release date so the product was available to the customer when it was promised.

The model was then presented to program management for use with upcoming programs but to little avail. It was viewed as too complex and not introducing any new factors to the program management process for estimating schedules. At this point the system test organization saw value in the model's insight into the test process and decided to continue using the model behind the scenes. The model's output was used to help test understand the structure of their process, to assess proposed program schedules, and to help educate program teams with respect to areas of risk. The output of the model was also used to educate the program teams on what was needed to improve products and reduce product validation time, but without providing them with visibility to the model.

Program schedules are based upon a set of assumptions that aren't typically quantified or documented, including the number of defects, number of test cells, the rate at which defects are fixed, and the duration of test cases. The PVM model served to document these assumptions in scenarios that were developed early in the planning phase of a program and then tracked throughout the validation process.

The PVM Model

This model (reference Figures 2,3,4) is based on the cumulative linear behavior for both defect discovery and fix rates, as well as test penetration rates. This simplification made the model easier to understand and more easily enabled measurement of the parameters. The model worked well doing what it was designed to do. It measured the critical parameters of the process and compared them to the assumed model values used in the scenarios to estimate the duration of product validation early in the planning stages of the design.

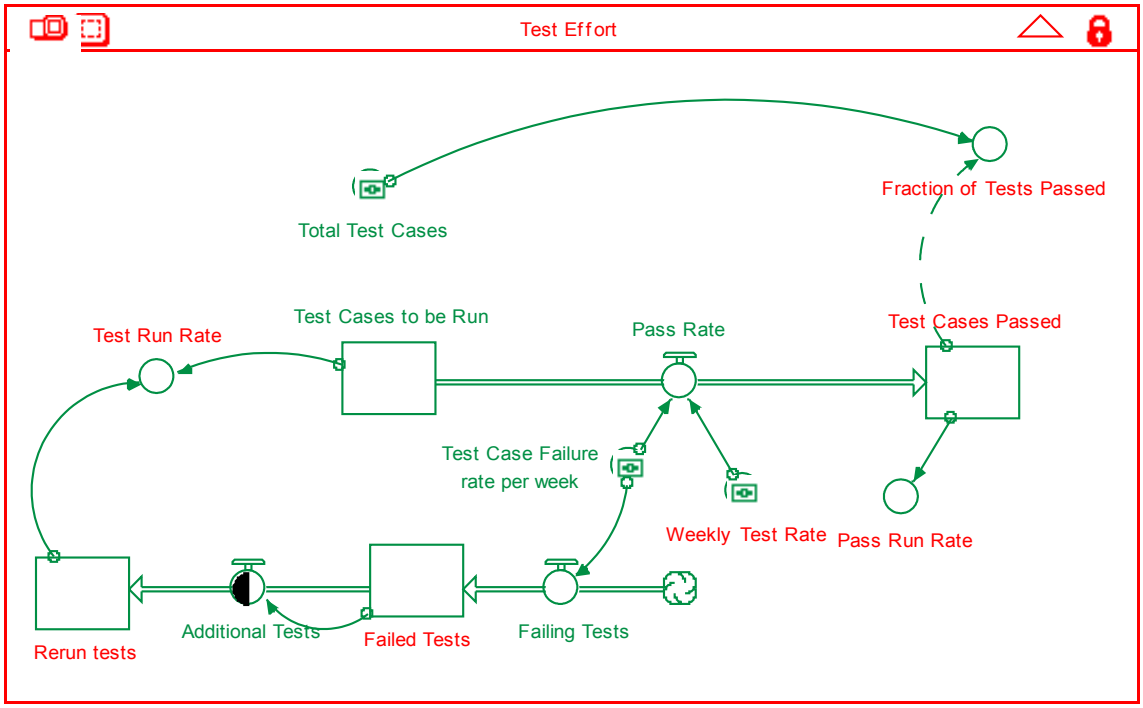

Figure 2

Figure 2 is the portion of the model that quantifies the product validation time with respect to the test effort. This portion of the model accounts for the number of test cases to be executed, the rate at which they run, and the rate at which they fail. The test rate was streamlined to a single value for the sake of simplicity. However, the test rate is dependent upon a number of factors, including the level of test automation, the duration of an executed test, the number of test cells, the number of defects found in the test effort, and the quality of the engineering documentation. Each of these factors is made up of additional factors. For example, the number of test cells is dependent upon the program budget and manpower allocated to test the product. Adding more detail for the sake of accuracy does not necessarily increase the value of the model. A more detailed example of the test rate model is shown in Figure 6 and discussed later in this paper.

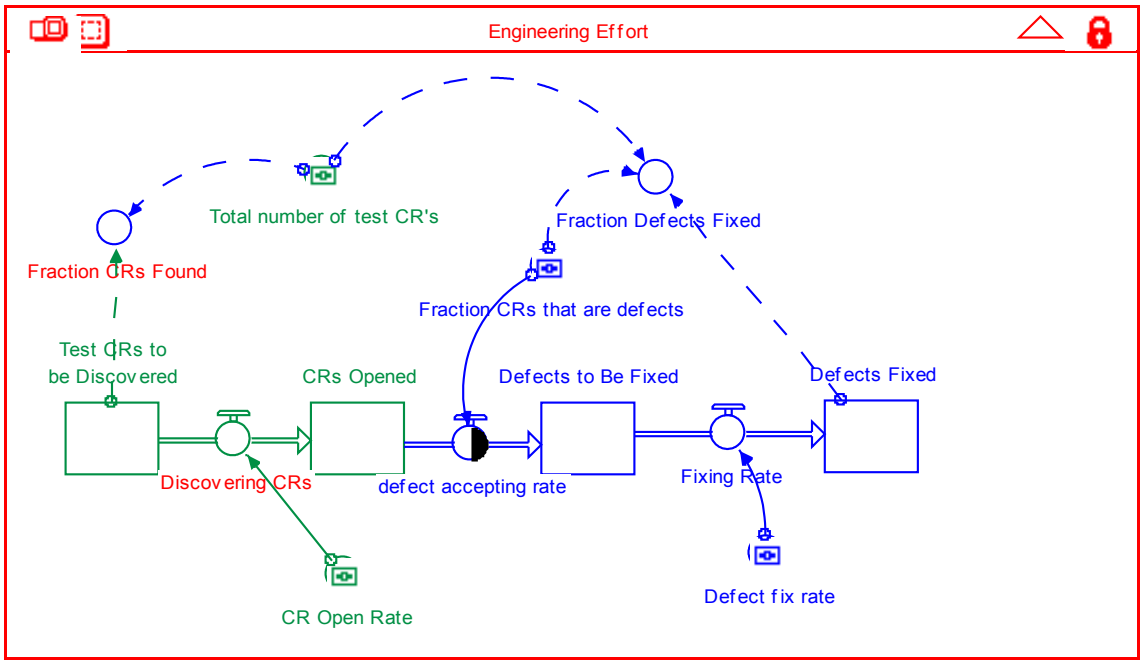

Figure 3

Figure 3 is the portion of the model that addresses the engineering factors having an impact on the product validation time. These factors include the number of defects that escape engineering ("Test CRs to be Discovered") and need to be found by system test, as well as the rate at which defects are found ("CR Open Rate") and fixed ("Defect fix rate"). The engineering section could also include the amount of engineering resources and the complexity of the design, but once again adding more complexity to the model does not necessarily make it a more useful model. For this modeling effort those additional factors and others are encompassed in the maturity level of the design as reflected by the "Test CRs to be Discovered" and the "Defect fix rate".

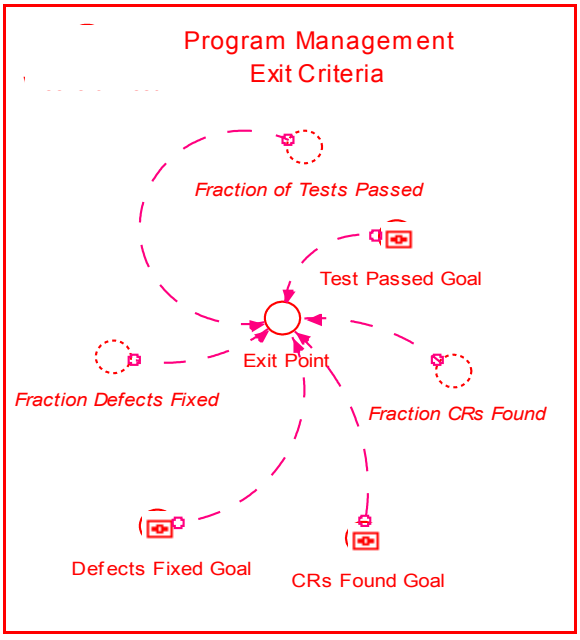

Figure 4

Figure 4 is the portion of the model that accounts for the program team's influence on the product validation time. This team is responsible for specifying the product's required level of goodness and functionality at the end of the product validation cycle. By making the exit criteria more or less stringent the duration of product validation can be adjusted.

Validating the Model

The stock and flow model was tested by mining historical and active program data from various test and defect tracking databases. This test and defect data was analyzed to compare model predicted validation times with actual validation times.

The PVM model validation was conducted against 20 programs with a resulting coefficient of determination (R^2) of 0.85. (Reference Figure 5 – the different colors represent different product families) This R^2 value indicates that if the correct parameters are entered into the model, the model has a high probability of accurately predicting the duration of product validation cycle. The difficult part is precise estimation of the parameters at the early stages of a program. To help select good starting values, historical data from similar product families may be used along with an understanding of the product differences. For example, is the program an iterative design change for a product or the introduction of a brand new technology? What product family does it belong to? Is it a new component product or a system integration of many components? How have the product family parameters behaved in the past?

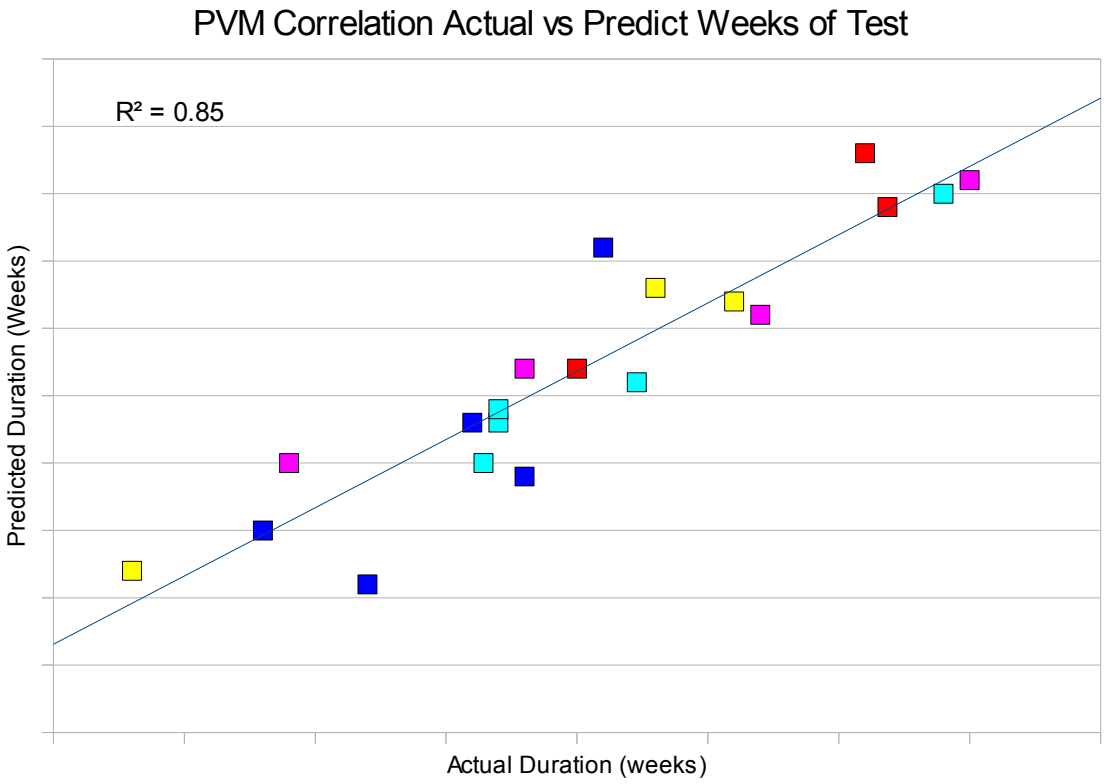

Figure 5

Use of the Model

Estimates of the PVM model parameters were used to build scenarios in the early program planning phases. Up to six scenarios were built for each program. As the program progressed through the validation process the parameters were measured and compared to the initial scenario parameter estimates. This iterative process provided insight regarding the value to which the parameters were converging. Thus the applicable scenario was identified as early as possible, giving an early indication of schedule risk, and identifying the reason for potential slippage so corrective action could be taken early in a program's life cycle. Tracking the parameters of the PVM model during product validation tended to remove human emotion from the decision-making process and provide the means to make data-based decisions.

The scenarios were developed in diverse ways to allow for different viewpoints. One set of scenarios began with the desired program schedule and the parameter values needed to achieve the schedule. These values were compared to their historical behavior to determine if the schedule was practical.

A second set of scenarios were built with the parameter estimates based on a product line's historical values. The model's estimated duration times were then used to establish system test input into the program schedule.

A third set of scenarios were assembled based on an optimistic view of the historical parameters. This set of scenarios was aimed at estimating the validation time if all the processes surrounding product development and test were perfect and the release cycle operated at maximum efficiency.

A major factor impacting the accuracy of the PVM model is the selection of a clear start date. Programs have many start dates such as the development start date, test start date, feature complete date, full functionality date, revenue-release candidate code date, and revenue release date. The amount of knowledge available depends on which start date is used. The later the start date the more predictive the completion date. The closer one is to the end, the more certain one is of where the end is. However, having accurate visibility to a completion date late in the program is not helpful in planning customer commit dates and doing resource planning. Earlier accurate predictions allow management more latitude in making contingency plans to meet schedules.

The PVM model tracking began as early as when products first underwent "early look" testing, if the test and defect data was being entered into their respective databases. Entry of early data did not always happen. Policies and agreements between test and engineering sometimes postponed logging early defect entries to prevent engineering from being overwhelmed with defects to sort due to the nature of the "early look" test. However, these same policies and agreements prevented program teams from having an early look at the critical parameters. To combat this problem, feedback loops are needed between the management team using the model and the testers entering the data into the databases.

Understanding a process's structure is key to its improvement. A common business quote is, "If you can't measure it you can't manage it." The PVM Model allowed the critical parameters to be measured so they could be managed.

In the course of using the PVM model various behavior conditions that hindered test's effectiveness became visible to test management so they became points to look for during the validation process:

- Engineering queuing up defects for fixing until the full feature set is coded.
- Products experiencing high volumes of defects.
- Poor product documentation.
- Early entry of products into test serving more for test becoming familiar with the product than for discovering bugs or penetrating the test cases faster.
- Collection of “early look” data (defects and test penetration) allowing for earlier validation of the critical parameters.
- If there are fewer than 50 defects then the test rate and number of test cases is the limiting factor, whereas if there are more than 75 defects then the lack of design maturity extends the validation period.
- The role that development engineering perceives for the test organization is important. If it is viewed as an extension of development then design maturity growth is slow. If final test is viewed as the customer then the product maturity growth is faster and the validation period is much shorter.
- Product validation time alone is a poor indicator of the product release schedule. The entire program phases need to be managed to reduce the release cycles.

The PVM model was used to formulate questions and discussions with program teams to help steer them toward improved product quality and schedule adherence. The first attempt to indirectly use the knowledge acquired to influence a program team resulted in the creation of a set of test entrance requirements reviewed with development engineering at the beginning of each program. Each of the items in the requirements list was there to improve test's efficiency and reduce test time:

- Development support (tools, code release plan...etc.),
- Test strategy (strategy for supporting early test, strategy for customer view... etc.),
- Metrics to be monitored and used for exit criteria,
- Training concerning expected operational and fault behavior, review of test cases, ... etc.,
- Requirements of the program and needs of test,
- Documentation concerning release note content, validation of customer facing documents, support for the development of procedure documentation.

Building a Complex Model (PVM2)

As System Test's experience with the PVM model grew it became obvious that there were more questions that needed to be answered.

- The initial model did not account for regression testing. This is testing where test cases are re-run to verify that the defects were fixed and the effort did not introduce any additional defects. How could the regression test policy best be defined to minimize the validation time?
- The discovery and fixing of defects does not occur in a linear manner. Would accounting for the non-linear behavior alter the predictive accuracy of the model?
- What impact does the quality of documentation have on the product validation time?
- If test automation were more aggressively instituted, what would be the benefit?
- How does the experience level of the testers impact the product validation time?

These questions led to the development of a more complex model that accounted for the non-linear behavior of defect discovery and fix rates and expanded the purpose of the model, using dimensionless multipliers to examine system test's strategic planning of the test environment. (reference Figure 6,7,8,9)

This model's intended purpose was no longer focused on estimating the duration of the Product Validation time but was rather aimed at assessing the impact of regression test policies, and conducting sensitivity analysis on process areas such as test automation, the quality of the Engineering documentation and the experience level of the testers.

The non-linear behavior of defect fix rates was modeled by introducing a parameter called "fraction of open defects fixed per week", which was based upon historical behavior of this parameter. (Reference Figure 9) The non-linearity of the defect discovery rate was modeled by introducing an initial failure intensity parameter similar to what is used in software growth modeling. This parameter was also based upon the historical behavior of previous products.

Dimensionless multipliers were added to the model (reference Figure 6) in the area of the test rate factors, to be used for test management discussions to understand the behavior of the process with respect to the changes in each of the following factors:

- Quality of documentation
- Experience level of testers
- Level of test automation (execution, recording and evaluation)

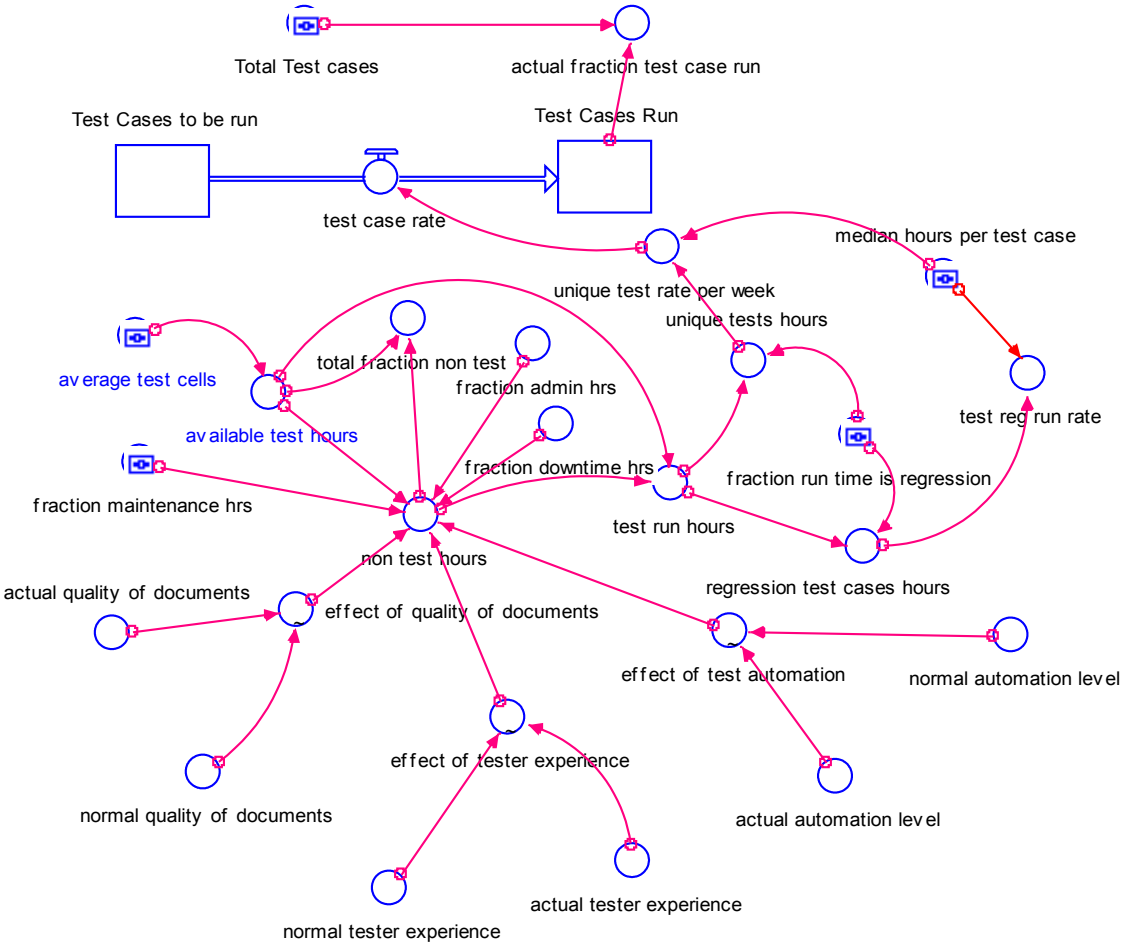

Figure 6

Figure 6 illustrates the Test Rate portion of the PVM2 model, which expands on the single "weekly test rate" parameter used in the model shown earlier in Figure 2. It uses dimensionless multipliers to model the "effect of quality of documents", "effect of test automation", and the "effect of tester experience". These multipliers were not always used in the model and when not in use were removed simply by setting the "actual" and "normal" levels to the same value. This model also quantified additional parameters that impacted the rate at which testing could be conducted such as the number of test cells, median duration time of test runs, and the amount of hours spent each week maintaining the test cells ("non test hours").

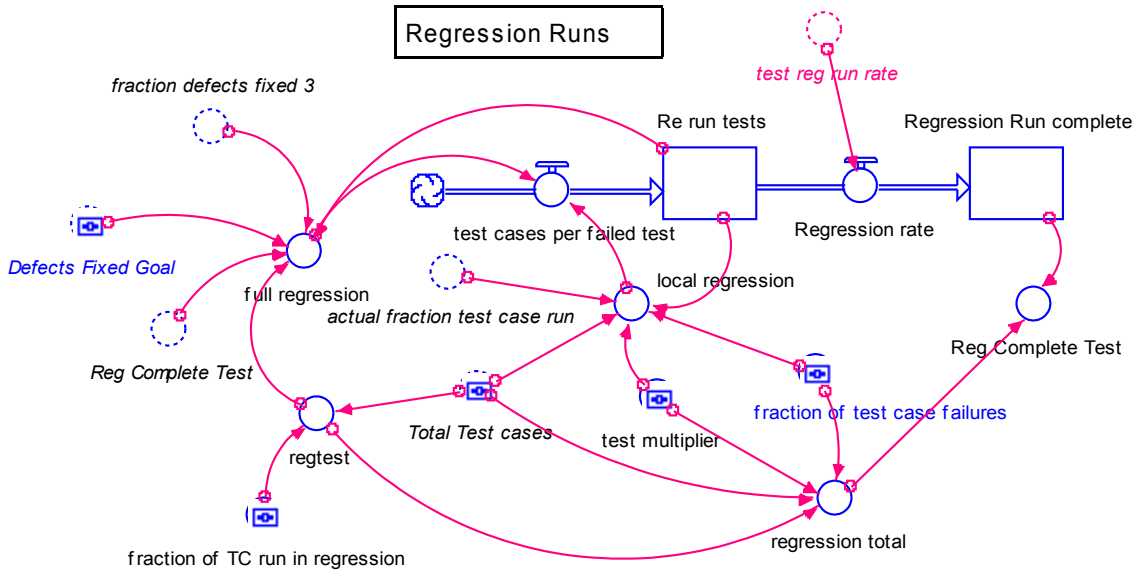

Figure 7

Figure 7 is the portion of the PVM2 model that accounts for the impact of regression testing. The regression testing could be local (a small set of tests re-run to verify code defects were fixed) or a full regression that is run at the end of the validation period as a final pass to the field. This portion of the PVM model used the "fraction defects fixed 3" to trigger the beginning of the final regression test run.

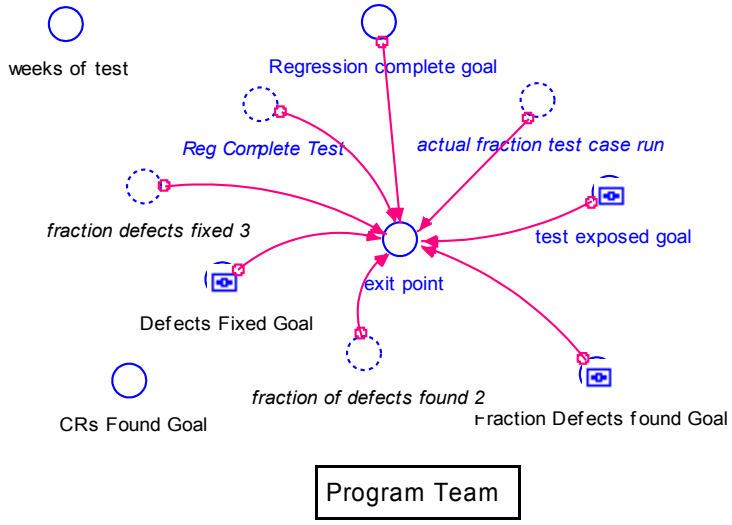

Figure 8

Figure 8 illustrates the program team controls that define the ship level quality of the product. This portion of the model is the same as the corresponding section of the PVM model with the addition of the "Reg Complete Test", as well as the "fraction of defects fixed 3", "fraction of defects found", and the "actual fraction test case run". Program teams have the ability to impact the duration of the Product Validation times by adjusting the level of risk they are willing to take concerning the quality of the product being sent to the customer.

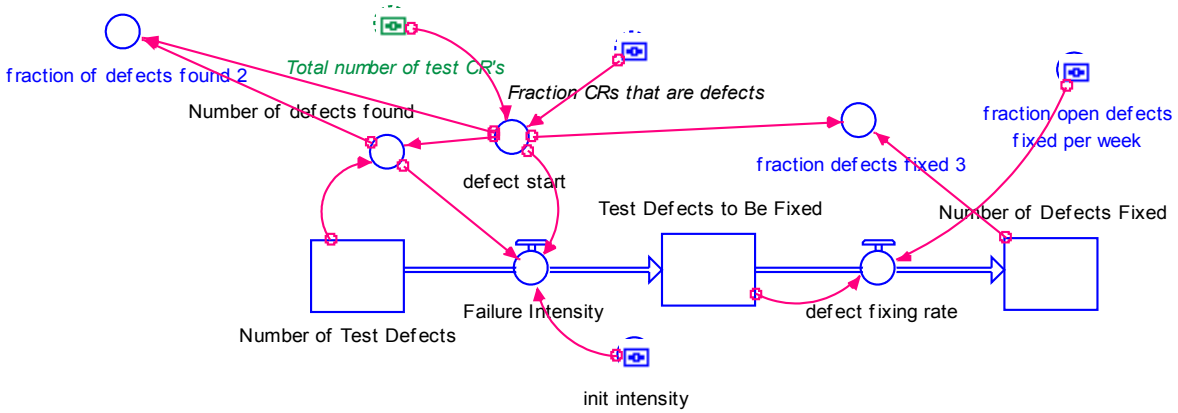

Figure 9

Figure 9 illustrates the section of the model used to describe the impact of defects on the Product Validation time. This section includes the non-linear discovery rate of defects based on the "init intensity" and their non-linear fix rate based on the "fraction open defects fixed per week".

This more complex PVM2 model could not be validated in the same manner as the initial PVM model, because of the inconsistent manner in which test cases were run and failure data was recorded. The boundary between the beginning of system test and the end of engineering development is obscure. Engineering/test agreements concerning when test results would first be entered into the respective databases are not uncommon.

The starting point for data recording has a significant impact on the non-linear distribution parameters. If the start time is not accurately displayed because it has been redefined, it appears that progress has been made when in reality nothing has changed. The effect of "soft" starts were not as much of a concern in the linear model since the defect discovery, fix and test case data was accumulated across the run time. When analyzing the cumulative data the point of most interest is not the beginning of the data but the end. With the non-linear model the start time influenced the initial intensity values which made this portion of the model very sensitive to the manner in which the start time was selected.

As well as the agreements that established the start time, testers are inclined to "explore" a product when it first enters test in an effort to validate test cases and understand the behavior of the product under test. These efforts are not likely to be recorded in a test database because of the uncertainty of the test output due to false starts and unstable test cases. It does not make sense to levy this extra work on the test environment just to increase the completeness of the data, especially when a simpler model works fine for the purpose for which it was intended. Only when critical factors are identified does it make sense to add more complexity to the definition of a model. This supports the desire to have models as parsimonious as possible.

The information obtained from the PVM2 model was used to challenge the test organization's perceptions of areas of risk and opportunity, especially in the areas where the dimensionless multipliers were used. It was also used indirectly in setting policy regarding an acceptable quality level of the documentation provided to test. Once again, a program can benefit from the modeling effort without having to view the model.

Increasing the model's complexity did not increase its predictive accuracy, but instead changed the purpose and use of the model, from a validation predictive tool to a means of assessing and

quantifying system test policies and strategies concerning the management of factors that influenced the rate at which testing could be conducted.

Additional Observations

When using data to qualify and/or operate a model it should be remembered that databases are inherently “dirty.” If the goodness of a model is dependent solely on the accuracy and cleanliness of the data then the model's usefulness will be limited or misleading. In addition. if heroic measures are instituted to insure absolute accuracy, then the data collection labor can be a waste of time and in itself become a major impact parameter.

The model may never reach a point of absolute accuracy, nor does it need to, since the object of testing is to execute tests that validate a new product and not spend large amounts of time entering data. From a management perspective the current level of accuracy was sufficient to answer the questions of interest.

Summary

The introduction of System Dynamics tools to the system test organization helped it better understand the structure of the product validation process and how it is influenced by endogenous as well as exogenous factors.

The main lesson learned concerned the magnitude of the product validation processes' complexity, and the number of factors that needed to be managed if indeed there was to be a reduction in product validation time. System test had direct control over such a small number of those influencing factors that there was little wonder that they had not been successful in reducing the validation time in past efforts. To reduce the product validation time would require changing the manner in which the entire program identified and managed the critical metrics. This is not new news, but the modeling did provide a means to quantify program assumptions, and the behavior of critical parameters during the program validation cycle as well as identify schedule risk.

This paper makes it sound like the problem of product validation duration was solved. After all, the critical parameters were identified and quantified and their interactions understood. All that remained was for managers to take action to manage these parameters and the duration of product validation would plummet.

However, the biggest challenge in this use of the system dynamics approach is the Limited Growth Archetype behavior that constantly needs to be managed. Within any company or organization there is constant change, people come and go, management and programs change, and companies reorganize or restructure. You cannot do a SD model, or any new process change for that matter, and expect it to sustain itself without continuous monitoring and educating. The model results need to be continuously fed back to the users of the model and to those creating and entering data that is used to support the modeling effort.

What has evolved is a data based means of estimating validation times and setting realistic schedules. While this experience did not cause engineering and program management to fully appreciate their impact on the product validation time, system test was able to ask a different set

of questions when communicating with the other members of the program team which helped guide the discussion to areas that supported the program's desire to reduce the schedule.

It is also worth mentioning that this effort also brought a new level of credibility to the test organization. Prior to the use of the SD approach it was not unusual for test input to be discounted. The understanding and insight the SD modeling effort provided allowed test to raise more creditable issues that were acknowledged as valuable input to the product release cycle.

The Wizard of Oz approach worked well when utilizing system dynamics within the system test environment and driving understanding of the behavior of the product validation process. However, to further advance the usage of the SD modeling in this context, work needs to be done to actively address the causes for the parameter behaviors. This will require extending the usage and education outside the system test area, to bring the model out from behind the green curtain, to reveal the wizard and integrate the engineering and program management disciplines so the critical parameters and interactions can be identified and more actively managed.

References

1. Forrester, Jay W., *Industrial Dynamics*, Cambridge: M.I.T. Press, 1969

2. isee systems, inc., *Applying Systems Thinking and Common Archetypes to Organizational Issues,* http://www.iseesystems.com/store/Training/ApplySysThink.aspx,

3. Sterman, John D., *Business Dynamics: System Thinking and Modeling for a Complex World*, Boston: McGraw-Hill, 2000

4. Fisher, Diana, *Modeling Dynamic Systems: Lessons for a First Course* , 2nd Edition, Acton: Creative Learning Exchange, 2005

5. Meadows, Donella, *Thinking in Systems,* Vermont: Chelsa Green Publishing, 2008