Software Development and Testing: A System Dynamics Simulation and Modeling Approach

KUMAR SAURABH
IBM India Pvt. Ltd.
SA-2, Bannerghatta Road, Bangalore. Pin- 560078
INDIA.
Email: ksaurab5@in.ibm.com, kumar.davv@gmail.com

Abstract: - Software-development and testing is a complex activity that often shows signs of contradicting instinctive activities, in that outcomes can vary drastically with deliberate consequences. Software-development and testing has many complexities, including dynamic behavior and feedback mechanisms, as well as various interacting factors. System dynamics is a modeling methodology that is well suited to explaining the root causes of contradicting instinctive activities — through its focus on building a simulation model that reflects causal relationships, feedback and delays. The production of a high quality software product requires application of both defect prevention and defect detection techniques. A common defect detection strategy is to subject the product to several phases of testing such as unit, integration, and system. These testing phases consume significant project resources and cycle time. As software companies continue to search for ways for reducing cycle time and development costs while increasing quality, software testing processes emerge as a prime target for investigation. This paper presents a system dynamics model of software development, better understanding testing processes. Motivation for modeling testing processes is presented along with a an executable model of the unit test phase. It motivates the importance of software cycle time reduction. The objective of the research is to provide decision makers with a model that will enable the prediction of the impact a set of process improvements will have on their software development cycle time.

Key-Words: - Cycle Time Reduction, Stocks, Simulation, Unit Testing.

1 Introduction

Measurement of both the product and development processes has long been recognized as a critical activity for successful software development. Good measurement practices and data enable realistic project planning, timely monitoring of project progress and status, identification of project risks, and effective process improvement. Appropriate measures and indicators of software artifacts such as requirements, designs, and source code can be analyzed to diagnose problems and identify solutions during project execution and reduce flaws, revision (effort, resources, etc.), and cycle time. These practices enable organizations to achieve higher quality products and reflect more mature processes, as delineated by the CMMI. Unfortunately, useful measurements related to the development of products coded to meet the requirements of secure software are in their infancy, and no consensus exists as to what measures constitute best practices. A review of the existing technical literature reveals the scarcity of *any* publicly reported, validated security measurements related to the software development life cycle. Nonetheless, there are some measures and practices used in software development that can be fruitfully extended to address security requirements.

2 System dynamics introduction

System Dynamics (SD) is a methodology whereby complex, dynamics and nonlinear interactions in social systems can be understood and analyzed, and new structures and policies can be designed to improve the system behavior. Similarly we can say, System Dynamics (SD) is a complex scientific and technological activity, for which is epistemological and methodological analysis could suggest some new and interesting perspectives both to practitioners and theorists of system dynamics (SD)[1]. The System models to have the most realistic representational content possible. There is a great different between purely Correlation or Statistical models and System Dynamics (SD. The System Dynamics (SD) models also try to offer explanation and understanding, not only forecasting and control.

2.1 Stocks and Level

These represent the accumulation of basic variables or quantities that change in example in a population model one stock may represent the population of a country.

2.2 Flow Variables

These variables represent the instantaneous flow rates. Unlike in physical systems where the rate variables mostly follow the laws of nature, in industrial and in many social and socio-economic systems, which are man – managed, rate variables often reflect overall policies governing individual decisions[2].

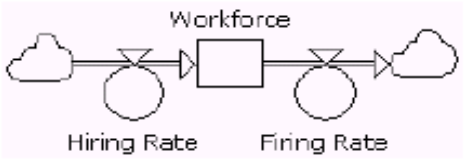

Fig 1 A simple model created in the graphical modeling language of Powersim Studio.

2.3 Connectors

A flow represents a physical link between stocks. However there are also information or dependency links.

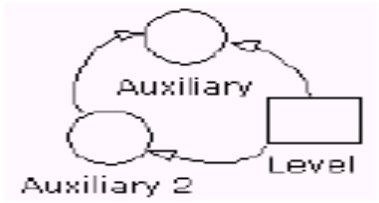

Fig 2 Information links connects various variables

3 Cycle time reduction model development

In order to illustrate the feasibility and usefulness of system dynamics modeling for process improvement assessment, we applied our approach to the software assessment process. For the purpose of our demonstration, we focus mainly on the question of cycle time reduction. We initially developed a base model corresponding to a typical organization's waterfall software development process. The software assessment model enables manipulation of a number of variables connected to the assessment process in order to understand their impact on software development cycle time.

Fig 3, however, does not reveal how time and manpower are allocated to perform each step in the assessment process, in order to keep the diagram and ideas presented simple. Each rate in Fig 3 requires that manpower be consumed in order to move work products from one step to the next. Fig 4 shows an incomplete, but representative implementation of the interface between the base model of the software development process and the process improvement model. Fig 4 represents the modeling of faults in the base process model of software development and illustrates the impact assessments have on fault production and fault notice in the base process model.

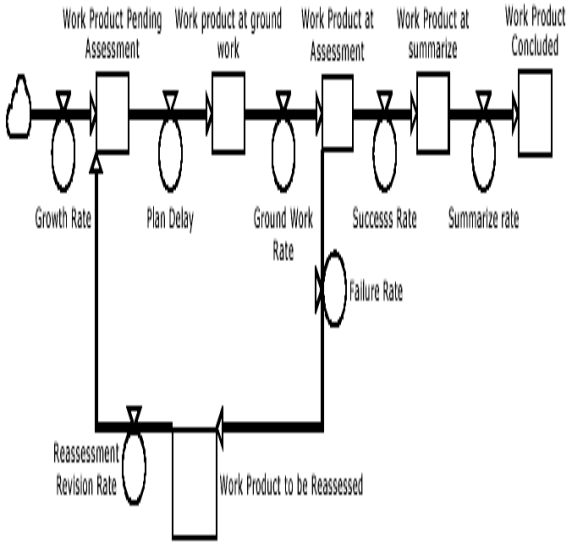

Fig 3. System Dynamics Model of Assessment Process

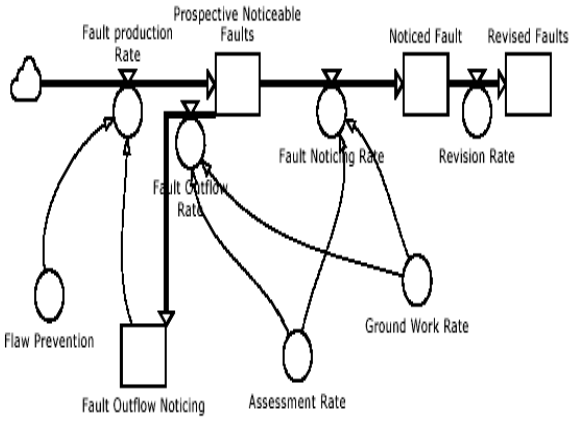

Fig 4 System Dynamics Model of the influence of Assessment Process on faults

3.1 Example model output

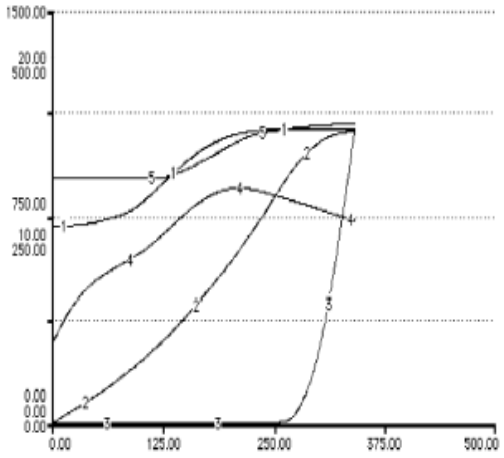

Fig 5. Assessment simulation Graph

Output from the simulator comes in two forms: numeric displays and graphs. Numeric displays show the current value of a simulation variable. Man-Days and Work Products Concluded are two examples of numeric displays. Graphs, on the other hand, display the value of simulation variables versus time. Each output curve is labeled with a number for ease of reading. There may be multiple units of measure on the vertical axis, each matched to the number of the curve it is representing.

The unit of measure on the horizontal axis is days. The five output curves represent: 1) currently perceived job size in terms of work products, 2)_cumulative work products developed, 3) cumulative work products tested, 4) total size of workforce and 5) planned completion date.

A demonstration of the use of the system dynamic models for predicting the cycle time reduction due to a process improvement is in order. Using the integrated model of the baseline waterfall growth life cycle and the software assessment process improvement, it will be shown how this modeling technique can be used for evaluating the impact that a proposed process improvement would have on growth cycle time.

The following demonstration is a simulation of a hypothetical software team employing the simple assessment model presented in this paper. The project being developed is estimated to be 64,000 lines of code requiring a total workforce of eight developers at the height of growth. Two scenarios of the project growth are simulated holding all variables fixed, except for the size of the assessment team and the percent of faults found during assessment.

Fig 6 is the output generated by executing the model with an assessment team size of six developers discovering 40 percent of the faults during assessment. When interpreting the graphical output, the story of the project is revealed. From Fig 6, the following story emerges. Curve 1, the currently perceived job size in work products, reveals that the project size was initially underestimated. As growth progressed, the true size of the project was revealed. Curve 5, the planned completion date, was not adjusted even as it became apparent that the project had grown in size. Instead, curve 4, the total size of workforce, indicates that the workforce was increased in size. In addition, though not shown on this graph, the workforce worked longer hours to bring the project back on e. Curve 2, cumulative work products developed, reveals that the project appeared to be back on plan, because there were no visible delays in growth of work products. It was not until system testing that problems in growth were discovered.

Curve 3, cumulative work products tested, reveals that system testing did not go as smoothly as expected. The poor performance of the assessment team pushed the notice of faults back to system testing. During system testing it was revealed that there was a good amount of revision to be done and as a result, the planned completion date, curve 5, was once again pushed back.

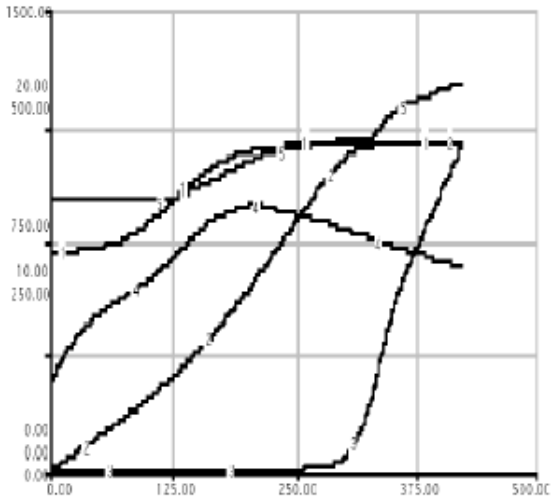

Fig 6. Software Assessment Graph 1

Fig 7 is the output generated by executing the model with an assessment team size of three developers discovering 90 percent of the faults during assessment. The story is much the same as that shown in Fig 5.

The big difference between Figs 6 and 7 is shown by curve 3, cumulative work products tested. Using more effective software assessments, this project was able to

discover faults early in the life cycle and correct them for much less cost than if they had been found in system test.
In addition, there were no major surprises in system testing as to the quality of the product developed.
Therefore, with no major amount of revision to be performed in system test, the project was able to finish close to its revised planned.

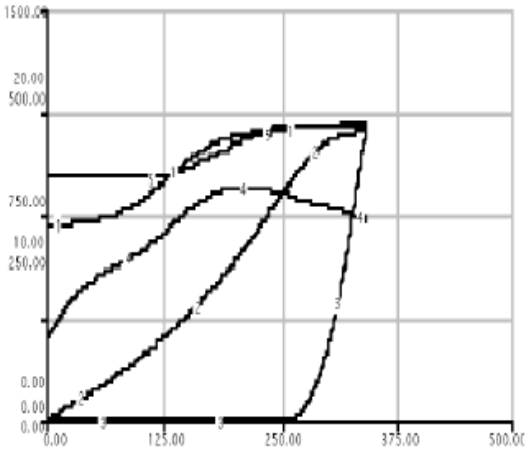

Fig 7. Software Assessment Graph 2

4 Modeling unit test phase

The controversy in the unit test phase revolves around the amount of unit testing that is performed. Although rigorous unit testing is recommended by many development standards, individual projects have been completed with various levels of unit testing dependent upon the other quality assurance tasks performed and the difficulty of creating a unit test environment. To investigate the impact of these various degrees of unit testing on software development cycle time, we developed a model of the unit test phase. This model assumes that the unit test phase begins after clean compilation and completes when the unit test criteria have been met and all defects have been fixed. It is important to note that we view the unit test phase as including both defect detection and repair. Repair consists of amending the code to remove the detected errors and retesting the code to verify the errors were removed. In order to model the impact of various unit test strategies, we also include a defect seepage cost in our model which addresses the cost of repairing defects missed by the unit test phase.

The basic inputs to our model are described below:

Variable Name	Description
Test volume	the volume of the unit test activity measured in lines of code to test
Test care	the care of the testing activity defined as the percentage of defects detected by the testing
Excellency of code	defined as the number of defects per KLOC which are detectable by the unit testing
Daily work force	the number of developers available for performing unit testing activities
Amendment efficiency	the number of errors fixed per developer-day
Cost to fix later	nber of developer-days needed in a later test phase to fix an error missed by unit testing

Table 1. Model Input Variables Description on Unit Test

The model outputs consist of:

able Name	Description
me for unit test	as the total number of days needed to complete the unit test phase
ost for unit test	as the total number of developer-days needed to complete the unit test phase
consequence	as the number of developer-days needed to repair the defects not detected during unit testing

Table 2. Model Output Variables Description on Unit Test

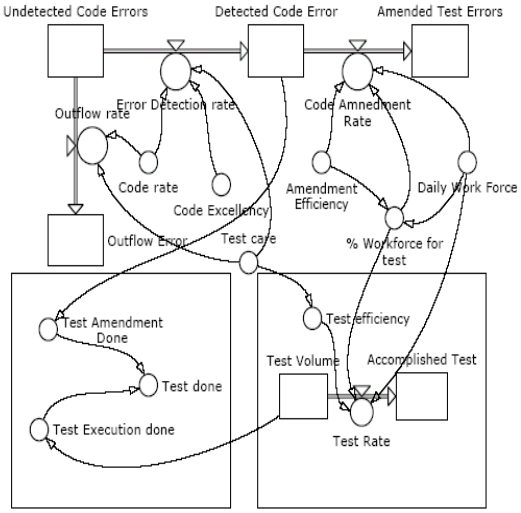

Fig 8 System Dynamics Model for Error Detection and Correction for Unit Test

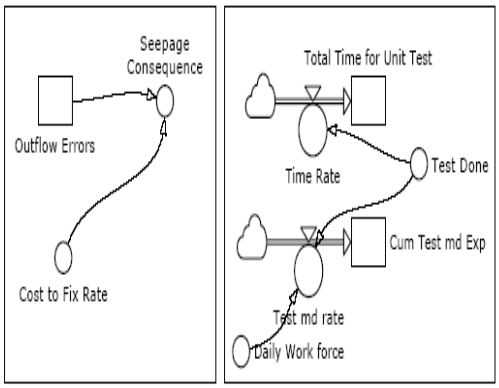

Fig 9 System Dynamics cost Model of Unit Test

A simplified view of our system dynamics model is presented in Fig 8 utilizing *POWERSIM VISUAL STUDIO-2005*. The model illustrates code errors being detected based on an error detection rate which is dependent upon the testing rate, the excellency of the code and the care of the testing. The care of the testing in turn affects the time needed to perform the testing. The model also illustrates the rate that detected errors are fixed which is dependent upon the percentage of developer time available for defect repairs, the number of available developers and the amendment efficiency. Defect seepage is also modeled along with the increased cost of repairing in later phases defects which were not detected by unit testing. To illustrate the kind of information which can be produced by this model, we extracted unit test data from an engineering organization. Three scenarios were executed with various levels of unit test care.

The levels of test care were:

Level	**ɔtion**
0.1	corresponding to very minimal unit testing
0.7	corresponding to a level of test care in which 70% of detectable defects were detected
1.0	corresponding to an idealized level of care in which all defects were detected.

Table 3 Levels Of Test Care On Unit Test

The results for each of the scenarios are presented in Table 4. To interpret the cost effectiveness of the unit test activity it is necessary to combine the columns for Total Cost for Unit Test and Seepage Consequence. For this particular organization's project scenario, the results indicate the benefit of reducing the unit testing effort.

1. **test volume** : 174,000 assembly equivalent lines of code
2. **excellency of code**: 0.39 defects per KLOC
3. **daily work force**: 5 developers available for performing unit testing activities
4. **amendment efficiency** : 8 errors fixed per developer-day
5. **cost to fix later**: 0.36 developer-days needed to fix an error missed by unit testing in a later test phase

The results for each of the scenarios are presented in Table 4. To interpret the cost effectiveness of the unit test activity it is necessary to combine the columns for Total Cost for Unit Test and Seepage Consequence. For this particular organization's project scenario, the results indicate the benefit of reducing the unit testing effort.
This can be explained by the low cost to fix a defect not detected during unit testing as determined by the metrics input to the model. Obviously these results will not

apply to all projects since variations of the input parameters will significantly alter the Total Cost for Unit Test and Seepage Consequence. For instance, when the cost to fix a defect not detected during unit testing is 1.0 errors per developer-day a test.

This can be explained by the low cost to fix a defect not detected during unit testing as determined by the metrics input to the model. Obviously these results will not apply to all projects since variations of the input parameters will significantly alter the Total Cost for Unit Test and Seepage Consequence. For instance, when the cost to fix a defect not detected during unit testing is 1.0 errors per developer-day a test care goal of 0.7 results in a lower overall cost.

Test Care	Total Time For Unit Test	Total Cost For Unit Test	Seepage Consequence
0.1	8.5	42.5	22.3
0.7	12.2	61.2	7.3
1.0	50.0	250.0	0.0

Table 4 Results of varying test care on Unit Test

5 Conclusion

This paper demonstrates to evaluate the effectiveness of process improvements. At this point in our work it gives developed a base model of the waterfall development life cycle and a process improvement model of software assessments. It can be enhanced for developing a base model of the incremental development process and creating a library of process improvement models. The model provides a framework for interpreting testing metrics and analyzing areas for optimizing testing processes. We are currently in the process of calibrating our testing model with actual industry metrics in order to provide projects with guidance on selecting their testing strategy.

References:

[1] Ashish Chand (2004), " Evolving ITES Capabilities", Business Process Outsourcing", Vol II An Indian Perspective ICFAI press Hyderabad, pp 91-97.

[2] Forrester, J W. 1985. "The" model versus a modeling "process". System Dynamics Review. 1(1&2): pp 133-134.

[3] J.P. Martino, "Technology Forecasting for Decision making" (2nd Edition)- American Elsevier, pp 43-47.

[4] Meade, N., "The use of growth curves in forecasting market development: A Review and Appraisal"/ of Forecasting 3,4 (Oct.-Dec. 1984), 429-451.