

Design and Sensitivity Analysis of a Formal Test Process Model

Praveen Ranjan Srivastava¹, Mahesh Prasad Ray²

^{1,2}Computer Science & Information System Group

Birla Institute of Technology and Science (BITS), Pilani, Rajasthan India

Email: praveenranjansrivastava,maheshray123@gmail.com

Abstract- Software testing is a quality-assess process for computer applications. Different qualitative and quantitative factors influence the process of testing during the software life cycle phase. In this paper a formal Test Process Model is designed and the change of behavior of the testing process is evaluated on the basis of several qualitative and quantitative influencing factors. The parameters are taken together, modeling is done and the higher order differential equation is solved by Euler's method. The change of the behavior of testing process is marked by changing individual parameters keeping the others constant.

Keywords - Software Testing, Software Life-Cycle, Modeling, Higher order differential equation, Euler Method.

I. INTRODUCTION

Software testing is an empirical technical investigation conducted to provide stakeholders with information about the quality of the product or service under test [1], with respect to the context in which it is intended to operate. It is one of the most time, effort and resource consuming stage of the software development life cycle [2]. So, it is always a demanding issue to control the influencing parameters so that the software test process can be controlled. The feature of a second generation process-centered software engineering environment, are grouped into three main areas: 1) process modeling, 2) process enactment, and 3) system architecture.

Previously, work has been done on formal and rigorous procedures applied to the software process [3][4][5], such as Statistical Process Control [6]. However, in practice, these and other less formal but rigorous approaches fall far short of the formalisms for process control that exist in other engineering disciplines. Here focus is on efforts on one significant phase of the Software Development Process (SDP), namely, the Software Test Process (STP). Though control of other phases of the SDP is often as important to an organization as the control of the test phase, the following two reasons motivated us to select the STP: 1) STP lends itself well to the characterization of input, output, and internal process variables and 2) there is a significant amount of data available from past and ongoing projects that is a key to the conduct of case studies to investigate the applicability of our model and approach. This second reason helps us to solve the difficult problem of identifying and estimating the key parameters to be included in any model of the STP.

II. THE STP CONTROL PROBLEM

Consider an application P under test. Assuming that the quality and schedule objectives were set at the start of the SDP and revised before the test process has started. For a scheduled target date or a set of checkpoints leading to a target date, the quality objective might be expressed in a variety of ways: 1) as the reliability of Product to be achieved by the target date [7], 2) as the number of errors remaining at the end of the test phase, or 3) as the increase in code coverage [8]. Further, the quality objective could be more refined and specified for each checkpoint. The experience of each tester [9] is another important factor to consider. It is the test team that carries out the testing activity and, hence, spends the effort that will hopefully help in meeting the objectives. The ever limited budget is usually a constraint to contend with. During the initial planning phase, a test manager needs to answer the question, how much effort is needed to meet the schedule and quality objectives? Experience of the test manager does help in answering this question. However, we approach this problem from a mathematical standpoint. The question stated above is relevant at each checkpoint. Assuming that the test manager has planned to conduct reviews at intermediate points between the start and the target date. The question might arise at various other points also, for example, when there is attrition in the test team. An accurate answer to the above question is important not only for continuous planning but also for process control and resource allocation. A question relevant for control is: How much additional test effort is required at a given checkpoint if a schedule slippage of, say, X percent can be tolerated? This question could be reformulated in many ways in terms of various process parameters

III. MODELING THE SOFTWARE TEST PROCESS

A number of variables and parameters are specific to the system test phase:

(W_f) – Effort in manpower

(S_c) – Software complexity

(PL) – Program level

(N) – Program length

(N_1) – Total occurrences of operators

- (N₂) – Total occurrences of operands
- (e_{et}) – Effective test effort
- (e_n) – The net applied effort
- (e_r) – Error reduction resistance
- (e_h) – Halstead effort
- (γ) – A constant characterizing the overall quality of the testing process
- (η₁) – Number of unique operators
- (η₂) – Number of unique operands
- (v) – volume of program in bits
- (t) – Time in days/weeks/months etc
- (r) – The number of remaining errors
- (ε) – The error reduction velocity constant

The model allows one to choose from a variety of a combination of existing complexity measures to obtain a value of s_c. For example, we could use program size, cyclomatic complexity [10], or a combination of both to compute s_c.

Model Assumptions

Four key assumptions are presented next. The model considers that these assumptions govern the STP.

Assumption-1 The rate at which the velocity of the remaining errors changes is directly proportional to the net applied effort (e_n) and inversely proportional to the complexity of the program under test [11][12]. This leads to.

$$r'' = e_n / s_c$$

$$\Rightarrow e_n = r'' * s_c \tag{1}$$

Assumption-2 The effective test effort is proportional to the product of the applied work force and the number of remaining errors [13][14][15]. This leads to

$$e_{et} = \zeta(S_c) * Wf * r \tag{2}$$

Where, $\zeta(s_c) = \frac{\zeta}{s_c * b}$ is a function of software complexity.

Parameter b depends on the characteristics of the product under test. The behavior of Assumption 2 is similar to the rate of decrease of errors [16], [17], [18] when software reliability models are applied to the STP [19].

Assumption-3 The error reduction resistance opposes, is proportional to the error reduction velocity, and is inversely proportional to the overall quality of the test phase [20][21]. This leads to

$$e_r = -\epsilon * 1 / \gamma * r' \tag{3}$$

for an appropriate constant ε. The negative sign indicates that the error reduction always opposes r'(t).

The justification of Assumption 3 relies on an analysis of its behavior under extreme conditions. A very low quality will induce a large resistance, i.e., as $\gamma \rightarrow 0, e_r \rightarrow \infty$. The same is true for values of r' i.e., the larger r' is, the larger the error resistance e_r. This implies that the faster one attempts to reduce

the remaining errors, the more likely one is to make mistakes that slow the test process.

Assumption-4 according to Halstead software science [22], the effort required to debug a program increases as the size of the program increases and the it takes more effort to implement a program at a lower level (higher difficulty) compared to another equivalent program at a higher level (lower difficulty). Thus the effort in software science,

$$e_h = V/PL$$

$$\Rightarrow e_h = \frac{(N_1 + N_2) * (\eta_1 + \eta_2) * \log_2(\eta_1 + \eta_2)}{2 * \eta_2} \tag{4}$$

From Assumptions 1, 2, 3, 4 we can obtain the Total Effort spent on testing a software product and monitoring the change of effort with respect to change in Time. Applying all these on Force equation F=m*a, where F is the total effort spent to make the product bug free and m is the number of errors of the project at the beginning phase of the testing.

IV. A DIFFERENTIAL EQUATION MODEL OF THE STP

Hence, e_n, e_r, and e_t can be related by the following force balance equation: Here all efforts applied to the system are on left-hand side and at the right-hand side mass is taken as the initial number of errors in the system [23]. i.e. say 100.

$$s_c * \frac{d^2 r}{dt^2} + \epsilon * 1 / \gamma * \frac{dr}{dt} + \zeta * w_f * r(t) + \{(N_1 + N_2) * (\eta_1 + \eta_2) * \log_2(\eta_1 + \eta_2)\} / 2\eta_2 = E_{total}$$

$$r(0) = 100 .$$

$$\Rightarrow \frac{d^2 r}{dt^2} + \epsilon * 1 / \gamma * 1 / s_c * \frac{dr}{dt} + \zeta * w_f * 1 / s_c * r(t) + \{(N_1 + N_2) * (\eta_1 + \eta_2) * \log_2(\eta_1 + \eta_2)\} / 2\eta_2 * s_c = K$$

$$r(0) = 100$$

V. SOLVING MODEL BY EULER METHOD

The Euler method may be considered the fundamental method for solving differential equations by numerical techniques [24],[25]. Its basis is the representation of differential equations by finite difference equations.

Consider the following general first-order differential equation:

$$\frac{d y}{d x} = f (x , y)$$

Applying the concept of finite differences (i.e.; the concept that differentials (infinitesimally small quantities) can be approximated by finite-sized quantities), we can write that

$$\frac{d y}{d x} \approx \frac{\Delta y}{\Delta x} = \frac{y_{n+1} - y_n}{x_{n+1} - x_n}$$

This statement becomes more nearly true as Δx becomes closer and closer to zero.

If we let h equal the step size such that

$$h = \Delta x = x_{n+1} - x_n$$

$$\text{Then, } x_{n+1} = x_n + h$$

Then the differential equation can be represented as,

$$\frac{dy}{dx} \approx \frac{\Delta y}{\Delta x} = \frac{y_{n+1} - y_n}{x_{n+1} - x_n} = \frac{y_{n+1} - y_n}{h} = f(x_n, y_n)$$

$$\text{Or } y_{n+1} = y_n + h f(x_n, y_n)$$

This equation, along with the equation $x_{n+1} = x_n + h$ becomes an iterative algorithm from which we can calculate a table of values of x_i versus y_i which represents the solution to the first-order differential equation. All we need is some starting conditions (also called initial conditions or boundary conditions) to get the algorithm started. Symbolically, the iterations would look like the following:

At,

$$x=x_0, y=y_0$$

⋮

$$x_{n+1} = x_n + h$$

$$y_{n+1} = y_n + h f(x_n, y_n)$$

Applying Euler's method we can simplify the Higher Order Differential Equation. First, the second order differential equation is written as two simultaneous first-order differential equations as follows.

Assume $X_1(t) = r(t)$

And $\frac{dX_1(t)}{dt} = X_2(t) = \frac{dr(t)}{dt}$

Again, $\frac{dX_2(t)}{dt} = \frac{d^2r(t)}{dt^2}$

But from Eq- I, $\frac{dX^2(t)}{dt} = \frac{d^2r}{dt^2} =$

$$[-[\epsilon * 1 / \gamma * 1 / s_c] \frac{dr}{dt} - [\zeta * w_f * 1 / s_c] r(t) -$$

$$(N_1 + N_2) * (\eta_1 + \eta_2) * \log_2(\eta_1 + \eta_2) / 2\eta_2 * s_c]$$

The above equation is in the form,

$$r''(t) = C_1 * X_1(t) + C_2 * X_2(t) + M$$

Where M

$$= -\{(N_1 + N_2) * (n_1 + n_2) * \log_2(n_1 + n_2) / 2n_2 * s_c$$

$$X' = \begin{bmatrix} X_1' \\ X_2' \end{bmatrix} = \begin{bmatrix} 0 & 1 \\ -\epsilon \frac{1}{\gamma s_c} & -\frac{\zeta w_f}{s_c} \end{bmatrix} * \begin{bmatrix} X_1 \\ X_2 \end{bmatrix} + M$$

Using Euler's method we get,

$$r_{i+1} = r_i + f_1(t_i, r_i, X_i)h \tag{Eq 2}$$

$$X_{i+1} = X_i + f_2(t_i, r_i, X_i)h \tag{Eq 3}$$

To find the value of $r(t)$ and since we are using a step size of 0.25 and starting At, $t = 0$, we need to take several steps to find the value of $r(t)$ with respective time units.

$$r(t) = e^{3-4.024\Delta t} (\eta_1 + \eta_2) + \text{Log}_e \left(\frac{w_f}{s_c} \right) + \frac{\epsilon}{\zeta \gamma}$$

For, $t = 0, t_0 = 0, r_0 = 100$ and Let, step size of $t, \Delta t = 1$.

VI. EFFECT OF PARAMETERS ON STP

Case 1

In the first case keeping the variables like $\eta_1, \eta_2, \zeta, s_c$ and γ [26] constant the effect of w_f on the testing process is shown below. The graphs (Figure 1) are plotted by taking the values from the table I. From the first graph it can be observed that as the initial number of errors is same in all the cases, at $w_f = 6$, after 35 days the test process is going to an end. Similarly from second, third and fourth graph it can be observed that at, $w_f = 5, w_f = 4, w_f = 3$ after 40, 50, 60 days the test process is going to an end respectively. Hence it can be concluded that increasing Test Effort in terms of workforce takes less time to make the product bug free.

TABLE I

Duration In Days	Number of Errors Remaining			
	$w_f = 6$	$w_f = 5$	$w_f = 4$	$w_f = 3$
0	100	100	100	100
2	84	88	92	95
5	70	74	80	85
10	52	56	60	66
20	26	34	40	44
30	9	11	13	21
35	2	7	9	11
40	2	2	5	4
50	2	2	2	3
60	2	2	2	2
70	2	2	2	2

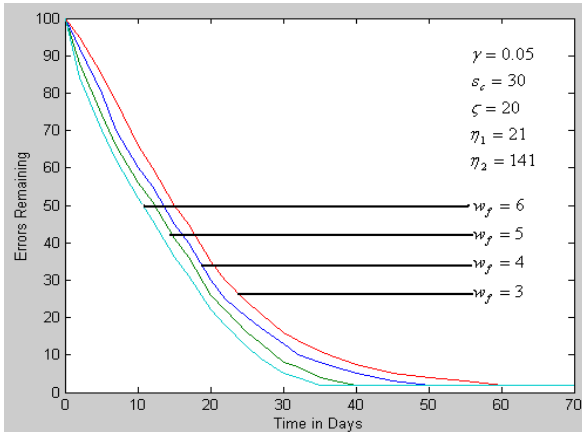


Fig. 1

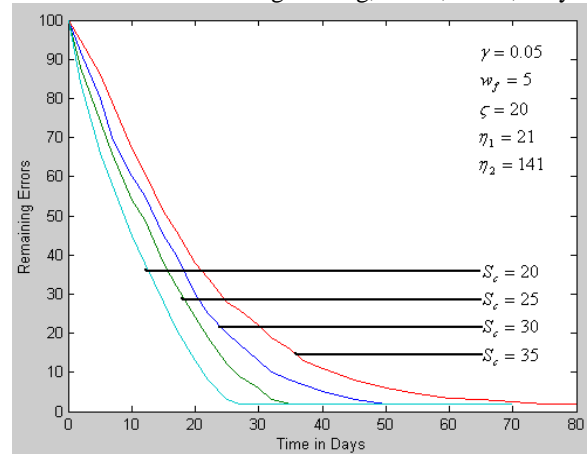


Fig. 2

Case 2

In the second case keeping the variables like $\eta_1, \eta_2, \zeta, w_f$ and γ constant the effect of s_c on the testing process is shown below. The graphs (Figure 2) are plotted by taking the values from the table II. From the first graph it can be observed that as the initial number of errors is same in all the cases, at $s_c = 20$, after 40 days the test process is going to an end. Similarly from second, third and fourth graph it can be observed that at, $s_c = 25, s_c = 30, s_c = 35$ after 50, 60, 70 days the test process is going to an end respectively. Hence it can be concluded that as the complexity of the product increases it takes more time to make the product bug free.

TABLE II

At, $\gamma = 0.05, w_f = 5, \zeta = 20, \eta_1 = 21, \eta_2 = 141$				
Duration In Days	Number of Errors Remaining			
	$s_c = 20$	$s_c = 25$	$s_c = 30$	$s_c = 35$
0	100	100	100	100
2	84	86	91	96
5	66	72	78	87
7	58	65	70	79
10	45	53	65	68
20	17	29	37	47
30	5	9	13	21
40	2	4	7	9
50	2	2	4	5
60	2	2	2	3
70	2	2	2	2

Case 3

In the third case keeping the variables like $\eta_1, \eta_2, \zeta, s_c, w_f$ and γ constant the effect of η_1 and η_2 on the testing process is shown below. The graphs (Figure 3) are plotted by taking the values from the table III. From the first graph it can be observed that as the initial number of errors is same in all the cases, at $\eta_1 = 21$ and $\eta_2 = 35$, after 40 days the test process is going to an end. Similarly from second, third and fourth graph it can be observed that at, $\eta_1 = 21$ and $\eta_2 = 721, \eta_1 = 141$ and $\eta_2 = 21, \eta_1 = 212$ and $\eta_2 = 721$ after 50, 60, 70 days the test process is going to an end respectively. Hence it can be concluded that as the complexity of the product increases it takes more time to make the product bug free.

TABLE III

At, $\gamma = 0.05, w_f = 5, s_c = 30, \zeta = 20$				
Duration In Days	Number of Errors Remaining			
	$\eta_1 = 21$	$\eta_1 = 21$	$\eta_1 = 141$	$\eta_1 = 212$
	$\eta_2 = 35$	$\eta_2 = 721$	$\eta_2 = 21$	$\eta_2 = 721$
0	100	100	100	100
2	83	87	93	95
5	61	72	80	85
7	52	67	70	78
10	37	51	61	67
20	14	25	37	52
30	4	13	13	23
40	2	3	7	8
50	2	2	3	5
60	2	2	2	3
70	2	2	2	2

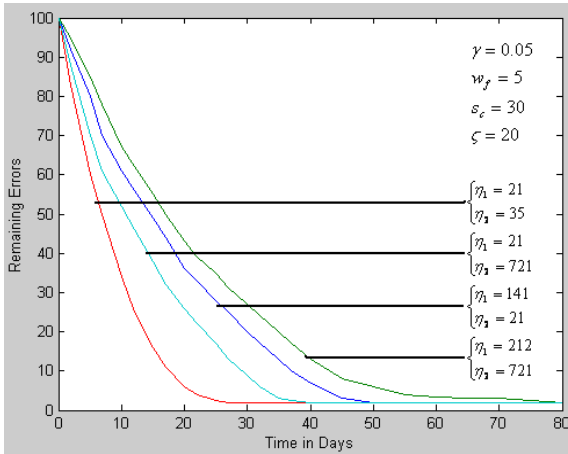


Fig. III

Case 4

In the fourth case keeping the variables like $\eta_1, \eta_2, \zeta, w_f$ and γ constant the effect of Quality on the testing process is shown below. Quality of Software Product $Q = 1/\text{Software complexity } (s_c)$ is considered. A number of Quality factors enter into a given organization, including:

1. Work force experience and expertise,
2. Test strategy/adequacy,
3. Coverage criteria,
4. Tool use/adequacy, and
5. Deadline pressure.

The graphs (Figure 4) are plotted by taking the values from the table IV. From the first graph it can be observed that as the initial number of errors is same in all the cases, at $Q=0.50$, after 40 days the test process is going to an end. Similarly from second, third and fourth graph it can be observed that at, $Q=0.40, Q=0.33, Q=0.30$ after 50, 60, 70 days the test process is going to an end respectively. Hence it can be concluded that as the quality of the product increases it takes less time to make the product bug free.

TABLE IV

At, $\gamma = 0.05, w_f = 5, \zeta = 20, \eta_1 = 21, \eta_2 = 141$				
Duration In Days	Number of Errors Remaining			
	Q=.50	Q=.40	Q=.33	Q=.30
0	100	100	100	100
2	84	86	91	96
5	66	72	78	87
7	58	65	70	79
10	45	53	65	68
20	17	29	37	47
30	5	9	13	21
40	2	4	7	9
50	2	2	4	5
60	2	2	2	3
70	2	2	2	2

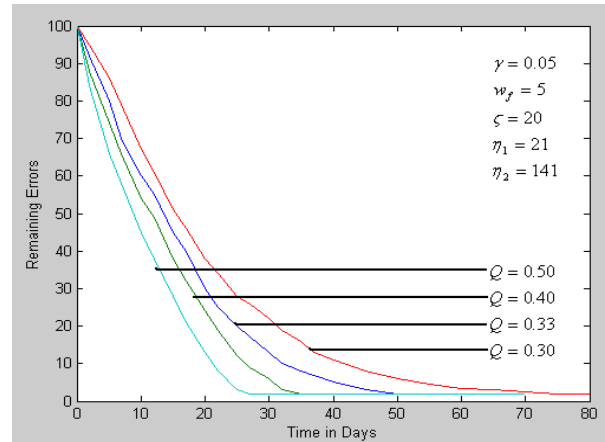


Fig. IV

VII. SUMMARY AND CONCLUSIONS

The Euler method of solving differential equations and the state variable approach is used to model the software test process. Different Qualitative and Quantitative parameters are taken and their influence on the software test process is observed by taking different case studies. This current work will help people within the organization to control the useful resources like work-force so that the process of testing can be achieved within limited period. Hence the work concludes that monitoring the test process along with identifying the major influencing parameters help to complete the testing process in a limited duration consuming less resources resulting profit in the software development.

REFERENCES

- [1] Exploratory Testing, Cem Kaner, Florida Institute of Technology, Quality Assurance Institute Worldwide Annual Software Testing Conference, Orlando, FL, November 2006.
- [2] Roger S. Pressman, Software Engineering: A Practitioner's Approach, McGraw-Hill, Sixth Edition, 2005.
- [3] S.L. Pfleeger, Software Engineering: Theory and Practice. Prentice Hall, 1998.
- [4] E. Kit, Software Testing in the Real World: Improving the Process. Addison-Wesley, 1995.
- [5] I. Jacobson, G. Booch, and J. Rumbaugh, The Unified Software Development Process. Addison-Wesley, 1998.
- [6] W.A. Florac and A.D. Carleton, Measuring the Software Process: Statistical Process Control for Software Process Improvement. Addison-Wesley, 1999.
- [7] J.W. Cangussu, R.A. DeCarlo, and A.P. Mathur, "Feedback Control of the Software Test Process through Measurements of Software Reliability," Proc. 12th Int'l Symp. Software Reliability Eng., Nov. 2001.

- [8] B. Marick, *The Craft of Software Testing*, Englewood Cliffs, N.J.: Prentice Hall, 1995.
- [9] Aditya P. Mathur, *Foundations of Software Testing*, Pearson Education, First Edition, 2008.
- [10] R.S. Pressman, *Software Engineering: A Practitioner's Approach*, McGraw-Hill, 1992.
- [11] T.H. McCabe, "A Complexity Measure," *IEEE Trans. Software Eng.*, vol. 2, no. 6, pp. 308-320, 1976.
- [12] A. P. Mathur, "A state variable model for the software test process," in *13th International Conference on Software and System Engineering and their Applications (ICSSEA'2000)*, vol. 2, (CNAM, Paris, France), pp. 1-10, CMSL CNAM, December 2000.
- [13] B.W. Boehm, *Software Engineering Economics*. Prentice Hall, 1981.
- [14] B.W. Boehm et. al., *Software Cost Estimation with Cocomo II*. Prentice Hall, 2000.
- [15] A. P. Mathur, "A state model for the software test process with automated parameter identification," in *In Proceedings of the 2001 IEEE Systems, Man, and Cybernetics Conference (SMC2001)*, (Tucson, Arizona), pp. 706-711, 10 2001.
- [16] K. Kanoun, M. Kaanicke, and J.C. Laprie, "Qualitative and Quantitative Reliability Assessment," *IEEE Software*, vol. 14, no. 2, pp. 77-87, Feb. 1997.
- [17] N.F. Schneidewind, "Measuring and Evaluating Maintenance Process Using Reliability, Risk, and Test Metrics," *IEEE Trans. Software Eng.*, vol. 25, no. 6, pp. 769-781, Nov./Dec. 1999. Cangussu Et Al.: A Formal Model Of The Software Test Process 795.
- [18] A. Wood, "Predicting Software Reliability," *Computer*, vol. 29, no. 11, pp. 69-77, 1996.
- [19] W.K. Ehrlich, J.P. Stampfel, and J.R. Wu, "Application of Software Reliability Modeling to Product Quality and Test Process," *Proc. Int'l Conf. Software Eng.*, pp. 108-116, 1990.
- [20] J.W. Cangussu, R.A. DeCarlo, and A.P. Mathur, "A Formal Model for the Software Test Process," Technical Report SERC-TR-176-P, Purdue Univ., Mar. 2001.
- [21] A. P. Mathur, "A formal model of the software test process," *IEEE Transactions on Software Engineering*, vol. 28, pp. 782-796, 8 2002.
- [22] Albrecht, A.J. and J. Gaffney, "Software Function, Source Lines of Code, and Development Effort Prediction: a Software Science Validation", *IEEE Transactions on Software Engineering*, SE-9 (6), 1983, p. 639-648.
- [23] G. Fulford, P. Forrester, and A. Jone, *Modeling with Differential and Difference Equations*. Cambridge Univ. Press, 1997.
- [24] S.R. Lay, *Convex Sets and their Applications*. John Wiley & Sons, 1982.
- [25] C.L. Lawson and R.J. Hanson, *Solving Least Squares Problems*. SIAM, 1995.
- [26] G. Cugola and C. Ghezzi, "Software Processes: A Retrospective and a Path to the Future," *Software Process Improvement and Practice*, 1999.