INCREASE RETURN on Investment of Software Development Life Cycle by Managing the Risk

—A Case Study

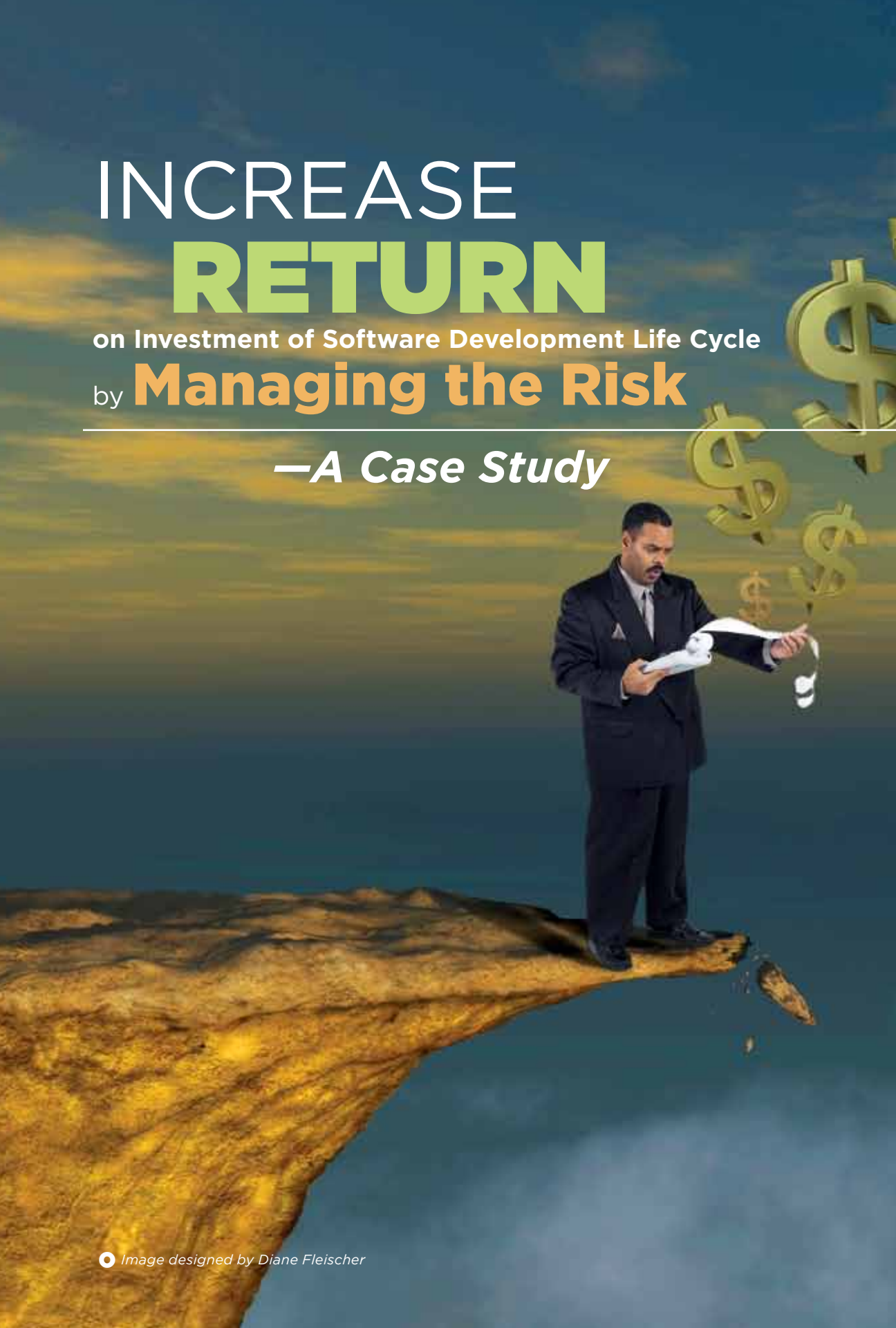


Image designed by Diane Fleischer

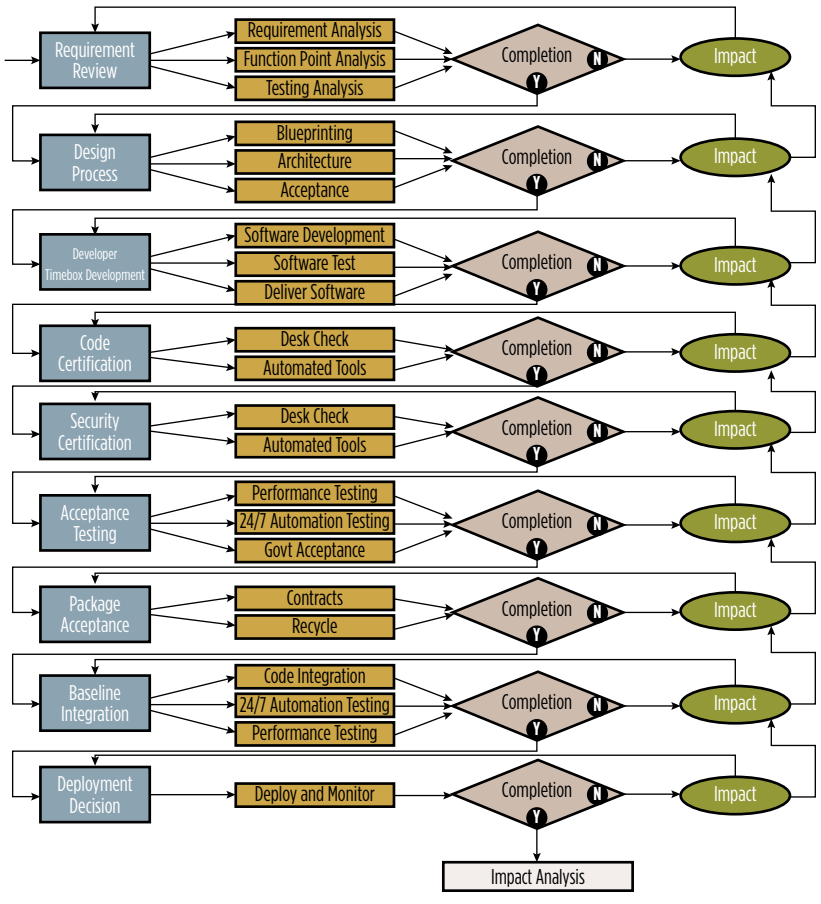
William F. Kramer, Mehmet Sahinoglu, and David Ang

This research article aims to identify and introduce cost-saving measures for increasing the return on investment during the Software Development Life Cycle (SDLC) through selected quantitative analyses employing both the Monte Carlo Simulation and Discrete Event Simulation approaches. Through the use of modeling and simulation, the authors develop quantitative analysis for discovering financial cost and impact when meeting future demands of an organization's SDLC management process associated with error rates. Though this sounds like an easy and open practice, it is uncommon for most competitors to provide empirical data outlining their error rates associated with each of the SDLC phases nor do they normally disclose the impact of such error rates on the overall development effort. The approach presented in this article is more plausible and scientific than the conventional wait-and-see, whatever-fate-may-bring approach with its accompanying unpleasant surprises, often resulting in wasted resources and time.

Keywords: *discrete event simulation (DES), Monte Carlo simulation (MCS), error or defect rate, return on investment (ROI), software development life cycle (SDLC)*

The science behind software development in metric terms of return on investment (ROI) is well known and taught by many. Much work has been accomplished in this area albeit lacking details of execution on a real-life problem (Ferreira, Collofello, Shunk, & Mackulak, 2009; Zhang, Kitchenham, & Pfahl, 2008; Zhang, Kitchenham, & Pfahl, 2010). The art of software development is a learned behavior and not one with which everyone becomes comfortable due to its intricacies and learning cycle. The same may be said with respect to software development life cycle (SDLC) management and distribution as depicted in Table 1, where the different phases of an SDLC process, when applied, provide specific inputs and expected outputs.

TABLE 1. LIFE CYCLE PHASE FLOW



Life Cycle Phase (Process) Flow

As with many processes, there is a beginning point and a delivery epoch. SDLC methodology is no different. It enables standardization for planning and organizing, and facilitates cost estimation. Though there are several different models available, many are tweaked to best fit the current process or a sequence of activities in a software development project. The life cycle used in this article (Table 1) has nine phases beginning with the requirement review and ending with the deployment decision. As one begins with the first phase (i.e., requirement review) and moves right, software developers will observe, at a minimum, the activities that must be performed in the phase (keep in mind this is a high-level depiction). Moving right, there is a decision to be made whether to proceed to the next phase or recycle back through the current phase for further refinement.

This decision is only one of many for the phases; however, it might be the most crucial. Not only will schedule and cost be impacted, but phase errors will drive substantial cost as well. An organization needs to understand the impact, and that is the intent of this article—namely to show the phase error impact to the SDLC, thereby reducing overall project management cost by improving the error rate.

Each phase will generate its own success criteria, allowing a development team to anticipate the degree of success that can be expected throughout the life cycle. Unfortunately, as a development team moves through the SDLC process, it is common to shift expected outputs to

the right and ultimately into the next phase, if only to remain on track regarding the end schedule or an expected financial burn rate. Ultimately, reality will set in and a price to be paid will become readily apparent, whether it be in the form of a scheduling or financial disruption.

This may be even more prevalent when it comes to the acquisition of custom software. To be better prepared for the impact of the shifting deliverables associated with the SDLC management process, one must understand the intricacies of the process and especially the impacts associated with a product that is either late or overbudgeted. Using a discrete event simulation (DES) and Monte Carlo simulation (MCS), as combined, may assist in quantifying a scenario impact. The primary raison d'etre of this article is to demonstrate the potential for modeling the SDLC management process and bring the cost-saving factor forward to improve the ROI by employing statistical simulation techniques. Therefore, the basis of this article is to bring attention to the use of modeling and simulation (M&S) in developing a quantitative analysis for discovering potential scheduling and financial ROIs within the parameters of meeting future demands of an organization's SDLC management process. More specifically, the potential impact is associated with errors accruing and accumulating throughout the process. That being said, one must be mindful that the methods used to compile this research article rely equally upon the art of simulation as well as the ever-enduring statistical and mathematical sciences behind the art of simulation. The statistical and mathematical computations used a significant amount of data gleaned from many years of software development experience. It is, however, through these years of experience with software development projects that we have come to appreciate an SDLC management process. Likewise, it is also during this process that we have learned to exercise

a degree of caution when evaluating those bidding to perform custom software development, who typically bid on the process with some degree of naiveté that views every requirement, algorithm, and interface as a nonissue and the work as always straightforward. Most of the time, this is not true, since hiccups invariably surface along the way, whether in the form of undefined requirements or bad test data. More often than not, unforeseen events occur, which ultimately impact both schedule and cost to the users' disadvantage.

Modeling and Simulation Methodology for a Case Study

To identify and incorporate software life-cycle phases along with function point analysis, software managers ought to associate the error rate per phase with the time distribution per phase. Organizations performing standard unit, integration, and functional testing will likely only remove approximately 70% of defects during the life-cycle phases (Jones, 2008). This practice will allow other defects to run through the life cycle until the bottleneck becomes apparent in the final testing phase. The model introduction takes this into account and assists with providing a rough order of magnitude (ROM) to the level of effort a program may encounter. In addition, the model also provides an alternative approach to facilitate ROMs with the appropriate schedule and additional resources.

Computer M&S, as programs or networks of computers mimicking the execution of an abstract model of many natural systems from physical and life sciences to social and managerial sciences, and primarily engineering, have become an integral part of digital experimentation. M&S proves useful to estimate the performance of complex engineering systems when too prohibitive for analytical solutions. A simulation is defined as the reproduction of an event with the use of scientific models. A model is a physical, mathematical, or other logical representation of a system, process, or phenomenon. Time-independent static MCS and, conversely, dynamic DES (to manage events in real time for engineering applications) have been extensively reviewed (Sahinoglu, 2013). Taxonomy-wise, simulated computer models may be stochastic or deterministic, and dynamic or static, and discrete or continuous. Computer simulation has been widely used in engineering systems to validate the

effectiveness of tentative decisions regarding a new plan or schedule, or its outcomes, without experiencing the actual conditions, which could cost more resources or partial to full destruction such as in the simulation of the nuclear bomb (Sahinoglu, 2007). In a book titled *Simulation Engineering* by Jim Ledin (2001), the author outlines his twofold purpose as follows:

> i) Simulation is an approach that can significantly accelerate the product development cycle and provide higher quality in the final system.
>
> ii) A simulation contains a set of mathematical models of one or more dynamic systems and the interactions between those systems and their environment. (p. 1)

Moreover, the Institute for Electrical and Electronics Engineers' *Spectrum* (June 2012) emphasized that the M&S effect is a creative and time-saving topic of interest relevant to automotive engineering of hybrid vehicles, finding solutions to treating nuclear waste, upgrading the nuts and bolts of the electrical power (Smart) grid, and supercomputing research, among other areas (Aoyama, 2012).

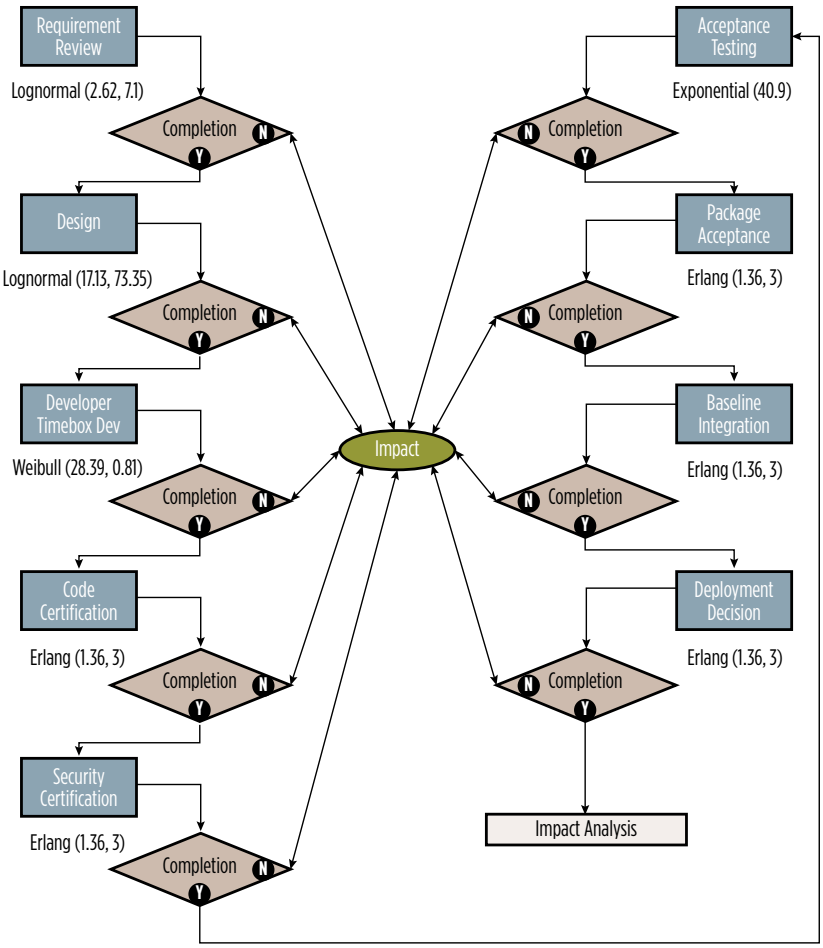
Simulation Approach

Table 2 depicts the conduct of an error rate analysis within the parameters of the SDLC management process. To better depict the probability distribution, Table 2 associates the probability distributions with each phase of the life cycle. Keeping in mind a waterfall model is in play, future research may requre further phase delineation among the many attributes of the phases. Note that:

- There is a need to simulate and model error rates within the SDLC process. Schedules and costs are impacted.
- Many models, such as waterfall, Agile, SCRUM, RAD, timebox, and spiral development methodologies exist today and could be used (Zhang, et al., 2010).
- This simulation model (Table 2) focuses on the error rates associated with waterfall methodologies.

- In order to determine cost per cycle and average cost per phase when using a development rate consisting of function point per staff month, calculate the error rates per phase and then aggregate with the suggested cost model.

TABLE 2. SOFTWARE DEVELOPMENT LIFE CYCLE DISTRIBUTION



Algorithmic Step-by-Step Approach Using Statistical Random Number Generation

Table 3 depicts iterations 1–1,000 and provides the details/samplings used in the simulation correlating the phases with probability distributions, the defect rates, repair rates, lambda, mu, standard deviation

TABLE 3. FACTS

Per Function Point: 1							Utilization Factor: 0.8			
	Phase 1	**Phase 2**	**Phase 3**	**Phase 4**	**Phase 5**	**Phase 6**	**Phase 7**	**Phase 8**	**Phase 9**	
	Requirement Review	Design	Code Development	Code Certification	Security Certification	Acceptance Testing	Package Acceptance	Baseline Integration	Deployment Decision	
	Lognormal	Lognormal	Weibull	Erlang	Erlang	Exponential	Erlang	Erlang	Erlang	
	2.62, 7.1	17.13, 73.35	28.39, 0.81	1.36, 3	1.36, 3	40.9	1.36, 3	1.36, 3	1.36, 3	
Iteration										
1	0.3149	0.4046	0.6248	0.0024	0.0792	1	0.0027	0.0792	0.0697	
2	0.3417	0.4051	0.0091	0.1525	0.057	1	0.1333	0.0119	0.0712	
3	0.354	0.4076	0	0.1211	0.1406	1	0.0775	0.0266	0.0959	
4	0.3106	0.3994	0	0.151	0.0567	1	0.0382	0.081	0.0653	
5	0.3404	0.4035	0	0.0737	0.101	0.09998	0.0347	0.039	0.0178	
998	0.3452	0.3985	0.9986	0.1153	0.0296	1	0.0163	0.0309	0.0522	
999	0.35	0.4053	0.001	0.1401	0.0463	1	0.073	0.1473	0.0663	
1000	0.2606	0.4051	0	0.1321	0.0492	1	0.073	0.1473	0.0663	
Avg Defects Per Phase (Mean)	0.3086	0.4024	0.2259	0.0681	0.0697	0.9809	0.069	0.0687	0.0704	2.2636
Std Dev	0.0443	0.0052	0.395	0.0464	0.046	0.0946	0.0456	0.0464	0.0451	0.7686
Days Per Phase	25	20	80	10	15	10	5	10	5	180
Avg Defects Per Day	0.012344	0.020121	0.002824	0.006807	0.004644	0.098088	0.013802	0.006869	0.014071	0.179571
Avg Repairs Per Day	0.01543	0.025151	0.00353	0.008509	0.005805	0.12261	0.017253	0.008586	0.017589	0.224463
Defect % Per Day	6.874	11.2052	1.5727	3.7908						
Aggregate	Lambda	0.1796								
	Mu	0.2245								
	Beta	1								
	Mean	2.2636								
	Std Dev	0.7686								

TABLE 4. FINDINGS AND EXCEL SPREADSHEET RESULTS

SINGLE TEAM				TWO TEAM			
Phases	**Probability of Waiting**	**Days per Phase**	**Days to Repair**	**Phases**	**Probability of Waiting**	**Days per Phase**	**Days to Repair**
P1	0.76	25	18.94	P1	0.23	25	5.67
P2	0.8	20	16.04	P2	0.23	20	4.55
P3	0.8	80	64.696	P3	0.23	80	18.54
P4	0.8	10	8.016	P4	0.24	10	2.35
P5	0.79	15	11.91	P5	0.23	15	3.5
P6	0.77	10	7.698	P6	0.22	10	2.24
P7	0.83	5	4.148	P7	0.23	5	1.14
P8	0.83	10	8.348	P8	0.22	10	2.2
P9	0.81	5	4.027	P9	0.23	5	1.14
Summation	7.19	180	142.82		2.05	180	41.32
Average	0.8				0.23		
Hourly Rate		$55				$55	
Team Members		10				20	
Hours/Work Day		8				8	
C_H = Cost Hourly T_M = Average Development Team Size D_H = Work Hours per Day D_R = Repair Days TC = Total Cost				C_H = Cost Hourly T_M = Average Development Team Size D_H = Work Hours per Day D_R = Repair Days TC = Total Cost			
$TC' = (((D_R \cdot D_H) \cdot T_M) \cdot C_H)$				**$TC' = (((D_R \cdot D_H) \cdot T_M) \cdot C_H)$**			
Single Team Total Cost $628,421.20				**Two Team Total Cost $363,633.60**			
SAVINGS $264,787.60							

(STD), and mean (Malone & Mizell, 2009). The average of the sampling was used along with a 180-day SDLC to determine defect rates per phase. These were used in the Java application to simulate and provide input to the findings in Table 4. Note the following:

- Function point count is maintained at one function point for the life-cycle period of 180 days. With the distribution per phase identified along with the days per phase, the Average Defects per Phase (ADP) is introduced with the summation of the ADP to be the average defect per one function point.
- Next, the Average Defects per Day (ADD) is calculated by dividing the ADP by the Days per Phase. This output becomes our lambda (λ) in the phase calculation in determining our Probability of Waiting (PoW).
- The Average Repairs per Day (ARD) is determined by multiplying the ADD by our utilization factor of a constant 0.8 (80 percent) from best practices (Malone & Mizell, 2009). This output becomes our mu (μ), also used in determining the PoW.

Results

Factors used to obtain results (Table 3) follow:

- Average Defects per Phase = (summation of each phase distribution)/iterations
- Days per Phase = variable set by experience
- Average Defects per Day = (Average Defects per Phase)/(Days per Phase)
- Average Repairs per Day = (Average Defects per Day)/utilization factor.

To make use of the facts in Table 3, a Java application (see Appendix, Java Source Code First Page) was developed to conduct several thousand runs for the simulation and ultimately provide a statistical summary to support Excel findings. The facts from the spreadsheet shown in Table 4 were placed into this homebrewed java application where the user can identify the inputs, the number of runs, and lastly, can run with either a single-team or a two-team simulation.

Table 4 represents only one screen shot with a single distribution, while arbitrarily using cost per hour of $55, team size of 10, and work hours per day to equal 8. One can vary the cost factors. Taking these factors into account, the cost formula in Equation (1) is as follows:

Total Cost = (Days to repair • work hours per day • team size • hourly rate). (1)

We can begin to readily determine that the errors per phase quickly outpace the efforts of a single developer and throw the schedule and cost model far to the right. However, by adding a second development team to assist with the fixing of the errors per phase, the cost and schedule are only slightly impacted (Malone & Mizell, 2009).

One can better appreciate the long-term impacts when dealing with contracts and why the lower bid may initially seem the best value; however, with the software development life cycle, this may not be the case. Improper preliminary analysis and use of resources could easily whirl the schedule and cost into an embarrassing tailspin. The core of this research precludes this handicap.

Other findings and Excel spreadsheet results highlighted in Table 4 follow.

- PoW is multiplied with the Days per Phase to obtain the Days to Repair for each specific team.
- Multiple variables are added to obtain realistic cost of software development teams (such as hourly rate of developers, team size, and hours per workday).
- The formula used for each team is: Total Cost = (repair days • work hours • team size • hourly cost).

Validation

Does the lowest dollar contract actually deliver the best value? This is what the research confirms positively.

Verification

Validation of error rates and function point rates came from Jones (2008).

Outcomes

Development teams can determine cost at granular phases within the SDLC as it pertains to error rates within software development. Upon running the simulation, the aggregated results show significant financial benefits. Factors used to obtain results are shown in Table 3 (Malone & Mizell, 2009).

Conclusions

The article responds to the following question: When required to analyze best-value contracts without using a simulation model, does the requestor actually obtain true cost by analyzing a single entity to develop software versus aggregated cost (Table 4) delivered from an additional pool of resources? Future work along with inputs from software development cost models will go a long way in producing a

> ***If the errors are identified in the early stages of a software development acquisition, contracting officers may be in a better position to avoid the lowest contract bid if they understand where proper resources, when applied, may actually decrease cost and schedule, thus delivering a successful acquisition and software functionality.***

better understanding of the true cost of software development and why there seems to be a schedule shift as the SDLC runs through its phases. This project scratches the surface by showing that the assumption by most software developers that all contracts and estimates provided are realistic, does not really portray the impact of errors to the schedules, which further increases cost. Some conclusive findings of interest are outlined below:

- Average cost per phase with single team to fix errors is an estimated $628,421.20 with the original summation of 180 days per phase.
- Adding an additional team to focus on errors, thereby increasing the cost for labor for two teams, equates to 42.14% savings. This is readily discerned in the reduced number of days to fix the errors. In fact, the second team will cost an estimated $363,633.60 in labor. The overall estimated savings is $264,787.60 for the cost of the repairs.
- Future and long-term analysis should focus on specific methodologies as well as on the coding language.

- Many organizations have invested in the use of the waterfall methodology and have been slow to appreciate the potential cost and schedule impact from error rates within the multiple phases of the SDLC.
- This article aimed to present a DES and MCS to determine an outcome that can be used to improve a process and cut costs. The error rate analysis project has done just that.
- Through lengthy discussions about rates within the MCS portion and the impact on business development systems, additional research and refinement may be sought to further develop the phase rates from within an organization. Additional research will provide better understanding of the impact for long-term software development and error rate impact.
- It is hoped that this and later work will enable future professionals in software development acquisition to establish a more definite cost analysis when confronting quantifiable data such as function points and development languages to give them a better understanding of the impact of development errors within the different phases of the waterfall SDLC.

An SDLC is a methodological process that from a high level can be used to determine schedules and costs and identify bottlenecks. However, it seems only recently that declining information technology budgets and increasing delivery costs require us to slice the life cycle into further granularity to understand better the cost and schedule impacts. In an attempt to correlate errors with phases and cost to fix, a prevailing assumption is that the cost of errors is flat. However, this may not be so. If the errors are identified in the early stages of a software development acquisition, contracting officers may be in a better position to avoid the lowest contract bid if they understand where proper resources, when applied, may actually decrease cost and schedule, thus delivering a successful acquisition and software functionality.

References

Aoyama, M. (2012). Computing for the next-generation automobile. *IEEE Computer*, 45(6), 32–37.

Ferreira, S., Collofello, J., Shunk, D., & Mackulak, G. (2009). Understanding the effects of requirements volatility in software engineering by using analytical modeling and software process simulation. *Journal of Systems and Software*, 82(10), 1568–1577.

Jones, C. (2008). *Applied software measurements: Global analysis of productivity and quality*. New York: McGraw-Hill.

Ledin, J. (2001). Simulation engineering—Build better embedded systems faster. Lawrence, KS: CMP Books.

Malone, L., & Mizell, C. (2009). Simulation model of spiral process. *International Journal of Software Engineering,* 2(2), 1–12.

Sahinoglu, M. (2007). *Trustworthy computing: Analytical and quantitative engineering evaluation*. Hoboken, NJ: Wiley & Sons.

Sahinoglu, M. (2013). Modeling and simulation in engineering. *Wiley Interdisciplinary Review Series (WIREs) Comput Stat 2013*, 5, 239–266.

Zhang, H., Kitchenham, B., & Pfahl, D. (2008). Reflections on 10 years of software process simulation modeling: A systematic review. Proceedings of International Conference on Software Process (ICSP 2008), *Lecture Notes in Computer Science* (Vol. 5007, pp. 345–356), Leipzig, Germany, May 10–11.

Zhang, H., Kitchenham, B., & Pfahl, D. (2010). Software process simulation modeling: An extended systematic review. Proceedings of International Conference on Software Process (ICSP 2010), *New Modeling Concepts for Today's Software Processes* (pp. 309–320), Paderborn, Germany, July 28–29.

Appendix

JAVA SOURCE CODE FIRST PAGE

```
//package negexp;
// W. Kramer

import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
import java.util.Random;
import java.text.*;

public class NegExp extends JFrame {

//elements of user interface
    private JLabel trialsJLabel;
    private JLabel meanJLabel;
    private JLabel devJLabel;
    private JLabel MuJLabel;
    private JLabel BetaServiceJLabel;
    private JLabel errorRateJLabel;
    private JLabel LambdaJLabel;
    private JLabel BetaJLabel;
    private JLabel servTimeJLabel; //package negexp;
// W. Kramer

import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
import java.util.Random;
import java.text.*;

public class NegExp extends JFrame {
```

Author Biographies

Mr. William F. Kramer has over 25 years in the Information Technology field. He has developed, sustained, and operated military information systems. His experience includes application design, development, software life-cycle management, and systems engineering. His education includes a BS in Computer Science from Chapman University, an MS in Management Science from Faulkner University, and an MS in Cybersystems and Information Security from Auburn University at Montgomery. Mr. Kramer is retired from the U.S. Air Force, with 20 years' active duty. He is currently employed with the U.S. Air Force as a federal civilian.

(E-mail address: wkramer@aum.edu)

Dr. Mehmet Sahinoglu is the founder/director of the Informatics Institute, and the Cybersystems and Information Security (CSIS) graduate program at Auburn University in Montgomery. As an Institute of Electrical and Electronics Engineers senior member and a Fellow of the IEEE Signal Processing Society, he authored 120 conference proceedings, 50 journal articles, *Trustworthy Computing* by Wiley (2007), *Cyber-Risk Informatics* by Wiley (2015), and managed 15 grants. Dr. Sahinoglu holds a PhD from Texas A&M and an MS from University of Manchester Institute of Science and Technology in Electrical and Computer Engineering, respectively.

(E-mail address: mesa@aum.edu)

Dr. David Ang is an industrialist and a business professor, with 20-plus years of experience in many facets of business from both a pragmatic perspective and real-life applications. He has published more than 70 articles in academic business research journals and conference proceedings. Dr. Ang holds a PhD in Industrial Management and Systems Engineering from the University of Alabama in Huntsville.

(E-mail address: dang@aum.edu)