

ESTIMATING DIRECT RETURN ON INVESTMENT OF INDEPENDENT VERIFICATION AND VALIDATION USING COCOMO-II

James B. Dabney
Systems Engineering Program
University of Houston – Clear Lake
Houston, Texas
dabney@cl.uh.edu

Gary Barber and Don Ohi
L3 Communications Titan Group
NASA IV&V Facility
Fairmont, West Virginia
{gary.barber, don.ohi}@L-3com.com

ABSTRACT

We define direct return on investment (ROI) as the ratio of reduction in development cost arising from early issue detection by independent verification and validation (IV&V) to the cost of IV&V. This paper describes a methodology to compute direct ROI for projects that don't maintain detailed cost-to-fix records. The method is used in a case study in which IV&V was applied to a mission-critical NASA software project. For this project, direct IV&V ROI was 11.8, demonstrating that IV&V was cost effective.

KEYWORDS

Verification and validation, return on investment, defect leakage, cost modeling.

1 Introduction

A standard management measure for determining the worth of an investment is return on investment (ROI), also known as benefit/cost ratio [1]. For software independent verification and validation (IV&V) [2], [3], we believe that there are many benefits and therefore many components of ROI. For example, benefits include reduced development cost, increased confidence in the final product, improved quality, reduced risk, and improved safety. Unfortunately, all of these benefits are difficult to measure. Consequently, IV&V ROI is inherently difficult to calculate. Among these benefits, reduced development cost is the least difficult to quantify. We refer to ROI based solely on reduced development cost as *direct ROI*. A previous paper [4] presented a methodology to compute direct ROI for projects that maintain detailed records of the cost to fix each discovered defect. This paper extends the methodology to the more common (in our experience) situation in which detailed cost-to-fix records are not maintained. The method exploits the COCOMO-II [5] model, calibrated to actual project results, to estimate cost-to-fix. The method is illustrated using a case study from a mission-critical

NASA software project. Additionally, the sensitivities of IV&V ROI to variations in IV&V scheduling and developer defect removal efficiency are studied.

We define direct ROI as the ratio $(C_x - C_i)/C_{IVV}$, where C_x is the project cost without IV&V, C_i is the project cost with IV&V, and therefore the difference $\delta C_r = (C_x - C_i)$ is reduction in development cost due to early issue identification by the IV&V team. C_{IVV} is the cost of the IV&V effort. Cost can be expressed in any consistent unit. Typically, equivalent person-months (EPM) or equivalent person-hours (EPH) are convenient.

The denominator of the ROI ratio is usually fairly easy to obtain from IV&V project records. The numerator, on the other hand, can only be estimated. While it is possible to determine the actual development cost, the cost savings due to early issue discovery and resolution cannot be known with certainty since it is not possible to know when (or even if) each issue identified by IV&V would have been found had IV&V not been used. Therefore, the central task in computing direct ROI is to devise a credible estimate of the cost savings.

The basis of our approach to computing δC_r is to compute rework cost for the actual with-IV&V data and to conservatively estimate the rework cost without IV&V by assuming that issues identified by IV&V would have been discovered later in the project by the developer with the same probability distribution as other issues discovered by the developer. In a previous paper [4], we considered projects for which actual rework costs are documented. For many projects, rework costs must be estimated using a tool such as COCOMO-II [5], [6].

The outline of this paper is as follows. First, we will summarize previous investigations related to IV&V ROI (presented in more detail in [4]). Next, we will review the direct ROI computation methodology [4]. Then, we will describe the modification to the direct ROI methodology using COCOMO-II formulas. We will then present results of applying this methodology to a NASA mission-critical software project. Next, we will discuss the sensitivity of

direct ROI to variations in IV&V scheduling and developer defect removal efficiency. Finally, we will summarize results and recommend future work.

2 Background

Although there has been extensive investigation over the past several years into the ROI of software process improvement [7], a rigorous methodology for measuring ROI of the various software assurance disciplines was not established prior to [4]. However, several earlier studies did shed light on IV&V ROI and provided valuable insight. The previous studies are discussed in some detail in [4]. They are summarized briefly here.

Arthur [2], [8] determined via a controlled experiment using two independent development teams, one of which employed IV&V, that IV&V has the potential to significantly increase the cost effectiveness of defect identification and removal. However, an earlier study at the NASA Software Engineering Laboratory [9] demonstrated that IV&V is not guaranteed to be cost effective, supporting the need to compute IV&V ROI so that IV&V resources may be used to greatest advantage. Rogers and McCaugherty [10] devised a rough estimate of IV&V ROI using defect removal costs from Jones [11] and actual error counts. Finally, Eickelmann [12] derived upper bounds for IV&V ROI based on developer Capability Maturity Model (CMM) level and IV&V budget. Together, the literature bearing on IV&V ROI supports three conclusions:

- The efficacy, and therefore the ROI, of IV&V can vary considerably.
- Employed properly, IV&V can be extremely beneficial, resulting in higher software quality and reduced cost to remove defects.
- No model had been proposed prior to [4] that used actual project cost and error data accumulated in active NASA projects to determine IV&V ROI.

3 Direct ROI Methodology

This section summarizes the direct ROI methodology presented in detail in [4]. The fundamental problem in computing IV&V ROI is to estimate the project cost without IV&V, given the project cost with IV&V and suitable project databases. The basis of computing the without-IV&V cost is the escalation of cost to fix an error as the project proceeds. Coupled with the probability distribution of developer discovery of the defects and actual cost-to-fix, without-IV&V cost-to-fix can be estimated.

3.1 Relative Cost-to-Fix Ratios

It is well-known that the cost to fix a software defect increases as the project proceeds. This fact has been recognized for many years [13] and is confirmed by recent data [14], [15], [16], [17]. This cost escalation is often used as a justification for software engineering

process improvements and software quality assurance activities [18], [19]. The escalation of cost to fix a defect has been reported by numerous sources. Based on analysis of cost-to-fix escalation data from [13] – [20], a normalized cost-to-fix escalation table (Table 1) was developed. Details of the derivation are presented in [4].

Table 1: Relative cost-to-fix ratios

Issue type	Phase issue found					
	Req	Des	Code	Test	Int	Ops
Req	1	5	10	50	130	368
Des		1	2	10	26	64
Code			1	5	13	37
Test				1	3	7
Int					1	3

The rows in Table 1 indicate the cost-to-fix escalation for each type of issue, assuming that all defects are introduced in the development phase corresponding to issue type. Therefore, the model predicts that the cost to fix a design issue discovered in the integration lifecycle phase is 26 times the cost to fix the same issue had it been discovered in the design phase.

3.2 Defect Leakage Probabilities

The defect leakage model is based on the assumption that the developer would discover the same percentage of defects existing at the beginning of a particular development phase in the absence of IV&V as they discovered with IV&V present. That is, the probability of developer discovery of a particular defect without IV&V present is the same as the probability of discovery actually experienced with IV&V present. Thus, the probability p_{tf} of the developer discovering a defect of type t in phase f is

$$p_{tf} = \frac{D_{tf}}{N_{tf}}$$

where D_{tf} is the actual number of defects of type t found by the developer in phase f and N_{tf} is the number of defects of type t known to exist at the beginning of phase f . That is, N_{tf} is the number of defects of type t actually found in phase f or a later phase by either the developer or IV&V. Thus, only known defects are counted because we have no credible estimate of unknown defects.

Next, in order to simplify the computations, a total probability P_{tif} is required for each defect type t , phase found by IV&V i , and development phase f . Here, P_{tif} is the probability that the developer would find a particular defect, actually found in phase i by IV&V, in subsequent phase f , computed by accounting for defect removal in previous phases:

$$P_{tif} = p_{tf} \left(1 - \sum_{\phi=R}^{f-1} P_{ti\phi} \right) \quad (1)$$

where R indicates the requirements phase and $f - 1$ indicates the phase before phase f .

3.3 Computing ROI

On projects for which the software developer tracks the cost to fix each defect corrected, it is necessary only to estimate the cost to fix each error identified by IV&V, had IV&V not been present. This estimate is simply the expected value of cost to fix each error given the cost escalation factors of Table 1 and probabilities P_{if} for the subsequent phases.

To illustrate the computation, assume that IV&V discovered a requirements issue in the design lifecycle phase. Using the cost-to-fix ratios of Table 1, the estimated cost to fix the error had IV&V not been present is

$$C_{fD} = (10P_{RDC} + 50P_{RDT} + 130P_{RDI} + 368P_{RDO})/5$$

$$c_x = c_i C_{fD}$$

where c_i is the actual recorded cost to fix the IV&V-discovered issue and subscripts R, D, C, T, I, O correspond to phases (and defect types) requirements, design, code, test, integration, operations, respectively.

The return on investment is the ratio

$$ROI = \frac{\sum c_x - \sum c_i}{C_{IVV}}$$

where C_{IVV} is the total IV&V cost.

4 COCOMO-Based ROI Computation

For many (in our experience, most) projects, the developer does not track the cost to fix each discovered defect. For these cases, it is necessary to estimate the cost-to-fix using a software cost model. This section describes the modification to the direct ROI methodology to estimate cost-to-fix using COCOMO-II [5] software cost estimating formulas.

4.1 COCOMO-II Calibration

COCOMO-II is a learning curve model which estimates development cost (equivalent person-months (EPM)) by

$$C_T = AS^E \quad (2)$$

where S is program size in source lines of code (SLOC) and A and E are system-dependent constants. Exponent E depends on five development project characteristics. The value of E for typical NASA projects is approximately 1.1. Coefficient A depends on seventeen key process areas which include management characteristics and software engineering practices.

To account for rework, COCOMO-II uses a term, BRAK, which is an estimate of the SLOC equivalent of the rework effort. The actual development cost (in EP months) and the delivered SLOC is normally available,

and it is possible to produce fairly accurate estimates of BRAK from issue logs and databases, as discussed in the next section. Given the delivered product size (new SLOC plus effective reused SLOC (ESLOC [5])), the effective project size is

$$SLOC = SLOC_{New} + ESLOC + BRAK \quad (3)$$

With exponent E estimated from project characteristics, coefficient A can be computed directly from Equation (2), thus accurately calibrating the cost model to the project results. Using this data to calibrate COCOMO-II to the project, we can estimate the without-IV&V BRAK and then compute a total without-IV&V development cost.

4.2 BRAK Estimation

Function points provide a means to associate the size of a software product with its functionality [21], [22]. A single unadjusted function point denotes a functional behavior of a software system. Function points are attractive because it is less difficult, early in a project, to estimate functional characteristics than to estimate directly SLOC. The function point methodology starts with characterization of the functional characteristics and classifying each by function point type. Next, each individual function point is multiplied by a scale factor k_w that depends on type of function point and complexity. This product is adjusted for development process characteristics, resulting in adjusted function points. Finally, adjusted function points can be multiplied by a language scale factor k_L that converts adjusted function points to SLOC.

The function point methodology can be used to estimate BRAK SLOC. To compute BRAK, the function points for each issue are assessed first. This is accomplished by reviewing each issue report and tabulating the number and complexity of each type of function point. Then, we compute the BRAK associated with each type of function point for each issue i as

$$BRAK_i = FP_i k_w k_L k_s$$

where FP_i is the number of function points of a particular type and complexity, k_w is the scale factor from [21] that depends on the type of function point and complexity, k_L is the language scale factor [5] that relates SLOC to FP for a particular programming language, and k_s is a scale factor that accounts for reduction in effort resulting from early issue detection. The basis for k_s is a requirements issue discovered in the integration phase, requiring complete rework for the particular requirement. Thus, for a requirements issue discovered in the integration phase, k_s is 1.0. Values of k_s computed directly from Table 1 are listed in Table 2.

Table 2: SLOC Reduction Factors k_s

Issue type	Phase issue found					
	Req	Des	Code	Test	Int	Ops
Req	0.008	0.038	0.077	0.385	1.000	2.831
Des		0.008	0.015	0.077	0.200	0.569
Code			0.008	0.038	0.100	0.285
Test				0.008	0.023	0.054
Int					0.008	0.023

Next, we must estimate the without-IV&V BRAK. For each function point type for each IV&V issue, the without-IV&V BRAK is computed from

$$BRAK = FP k_w k_L k_{sD}$$

where all terms are as previously defined except that k_s is replaced by k_{sD} which is an average of k_s for the remaining phases weighted by the percentage of developer-discovered issues per phase, in a manner identical to that used to compute c_x . Thus, k_s is project-independent and k_{sD} is project-dependent. Using the without-IV&V BRAK, effective without-IV&V project size is computed using (3) and estimated without-IV&V development effort is computed using Equation (2) (and the previously determined value of A). Finally, ROI is computed as the ratio of development cost reduction due to IV&V to the cost of IV&V,

$$ROI = \frac{C_x - C_i}{C_{IVV}}$$

where C_T is the without-IV&V development cost (estimated), C_i is the actual development cost experienced using IV&V, and C_{IVV} is the cost of the IV&V effort.

Note that for the with-IV&V case, we compute BRAK using both developer- and IV&V- discovered issues. For the without-IV&V case, we recompute BRAK only for IV&V-discovered issues. BRAK for developer-discovered issues remains the same and thus the increment to BRAK is due exclusively to IV&V-discovered issues.

5 Case Study

The COCOMO variant of the direct ROI methodology was applied to a moderately-sized software development project for a mission-critical, safety-critical near-real-time software system. The project entailed approximately 78,000 source lines of code (SLOC), including 30,000 lines of reused code. Total development effort (including rework) was approximately 381 EPM and the IV&V effort was approximately 53 EPM. This project did not track the cost to fix each issue, so it was necessary to use the COMOMO-II variant of the direct ROI model.

Table 3 lists the adjusted function points of all issues identified uniquely by IV&V. That is, an issue was credited to IV&V only if the developer did not also

discover the same issue in the same development phase. Table 4 shows the defect adjusted function points for the developer. Using the methodology described in Section 4, ROI was computed to be 11.8.

Table 3: IV&V-discovered issue adjusted function points

Type	Phase Found				
	Req	Des	Code	Test	Int
Req	637	318	62	57	20
Des		388	0	162	38
Code			284	23	68
Test				258	0
Int					0

Table 4: Developer-discovered issue adjusted function points

Issue type	Phase found				
	Req	Des	Code	Test	Int
Req	956	240	62	57	20
Des		293	0	162	38
Code			284	23	68
Test				258	0
Int					0

6 Sensitivity Analysis

In [4] the sensitivity of the direct ROI methodology to variations in cost-to-fix escalation was considered. Another important factor in IV&V ROI is the timing of issue discovery. The timing has two implications. The first is a direct consequence of the cost-to-fix escalation. The potential ROI impact of a particular issue is clearly greater the earlier in the lifecycle the issue is discovered. However, the timing of issue discovery (by IV&V and the developer) also affects the developer defect discovery probability distributions. To understand direct ROI consequences of variations in defect discovery phasing, a second sensitivity analysis, discussed next, was performed. The sensitivity analysis considered variations in IV&V defect detection timing and developer defect discovery timing.

6.1 Variations in IV&V Defect Detection

It is apparent that early lifecycle IV&V activities have the potential to produce higher direct ROI than late lifecycle activities. This component of the sensitivity study measured this effect by varying IV&V issue discovery rates across the lifecycle, based on the

reasoning that IV&V issue discovery rates will correlate with IV&V effort distribution. To test the sensitivity of ROI to the placement of IV&V effort, defect discovery profiles were generated for four cases for the same hypothetical project:

1. FULL: IV&V applied over the entire lifecycle
2. EARLY: IV&V applied only to the earlier lifecycle phases
3. LATE: IV&V applied only to the later portion of the lifecycle
4. NO DESIGN: No developer or IV&V defects discovered during the design phase (for the case where the developer skips the design phase)

In order to calculate the ROI for IV&V in each of these cases, defect discovery profiles were needed for both the developer and IV&V. Table 5 lists project characteristics that were held constant for all projects.

Table 5: Simulated project characteristics

Project Characteristics	
New SLOC	100,000 SLOC
Developer Effort	1,000 EPM
SLOC Conversion Factor, K_L (C Language)	128 SLOC/AFP
Average Adjusted Function Points per Defect	5.5 AFP/defect
Defects Introduced per Type ¹	
Requirements	156 defects
Design	194 defects
Code	274 defects
Test	200 defects
Integration	120 defects

Defect discovery per phase for both developers and IV&V was simulated by assuming a constant Defect Removal Efficiency (DRE) across phases for each issue type. DRE represents the fraction of issues present that were detected by either the developer or IV&V. The details of the simulation are provided in the sensitivity spreadsheet Table 6 lists the results from the sensitivity study of ROI to IV&V issue detection distribution.

Table 6: ROI Sensitivity to IV&V Issue Detection Distribution

Test Case	ROI
FULL	8.5
EARLY	14.7
LATE	10.9
NO DESIGN	12.8

The Early case resulted in the highest ROI, as expected. Defects detected early by IV&V would have the potential to leak the farthest had IV&V not been present.

This case also resulted in the highest number of defects leaking to the operations phase.

A surprising result was that the Late IV&V resulted in a higher ROI than Full lifecycle IV&V. The Late case provided better protection from leakage to operations than the early case. The reason for higher ROI than full lifecycle IV&V is the leakage of IV&V discovered defects for the case without IV&V all occurred in the steeper portion of the escalation curves. The no design phase case had a combination of the effects of the early and late IV&V cases.

6.2 Variations in Developer Defect Detection

To examine the impact of developer defect detection on the results of the direct ROI model, we used the Full lifecycle case from the previous experiment on variations in IV&V defect detection. For this experiment, we held the IV&V defect detection profile constant instead of allowing it to vary as defects present per-phase times the DRE. This approach was used to understand the effects on direct ROI of variations of developer defect profiles using identical IV&V results. The total number of defects detected by the developer was held constant across all of the cases to examine the effects of phasing independent of the number of defects.

Since the full lifecycle case from the experiment in section 6.1 was based on defect detection using a constant DRE, we consider that representative of a full lifecycle focus by the developer on defect removal and call it DEV FULL here. For the DEV EARLY case, the developers detect ~ 90 % of their portion of the defects in phase. For the DEV LATE case, the bulk of the defects discovered by the developer were in the test, integration, and operations phases.

Table 7: ROI sensitivity to developer issue detection distribution

Test Case	ROI
DEV EARLY	12.5
DEV FULL	8.5
DEV LATE	11.6

Table 7 shows that the phasing of developer issues does have an impact on the direct ROI results. The impact is due to the dependence of direct ROI on developer defect discovery probabilities. That the DEV LATE case results in higher IV&V ROI is easy to understand – delaying the developer defect discovery activities increases the probability of finding defects later in the lifecycle. The DEV EARLY case increases direct ROI because issues discovered in-phase by the developer don't contribute to the probability computations.

7 Conclusions and Future Work

The direct ROI methodology provides a straightforward means to compute direct ROI for IV&V projects. This paper has presented a variant of the direct

¹ Defect data based on [23]

ROI methodology that uses the COCOMO-II formulas to estimate rework costs. The use of the methodology was demonstrated using a case study and produced results similar to those achieved previously for a project for which detailed cost-to-fix records were maintained.

The sensitivity analysis of this paper demonstrated that the direct ROI model is moderately sensitive to variations in the timing (with respect to the development lifecycle) of IV&V and developer defect detection activities.

8 Acknowledgements

This research was supported by the NASA Independent Verification and Validation Center and L-3 Titan Group.

References

- [1] W. G. Sullivan, J. A. Bontadelli, and E. M. Wicks, *Engineering Economy, 12th Ed.*, Prentice Hall, Upper Saddle River, New Jersey, 2002.
- [2] J. D. Arthur, W. Frakes, S. Gupta, M. Cannon, M. K. Groener, Z. Khan, *A Study and Project-Based Evaluation of the Software Engineering Evaluation System SEES*, Technical Report, Department of Computer Science, Virginia Polytechnic Institute and State University, Blacksburg, Va., 1997.
- [3] S. Easterbrook, The role of independent V&V in upstream software development processes, *2nd World Conference on Integrated Design and Process Technology*, Austin, Texas, 1996.
- [4] J. B. Dabney, G. Barber, and D. Ohi, "Estimating Direct Return on Investment of Independent Verification and Validation," 8th IASTED Conference on Software Engineering and Applications, Cambridge, Massachusetts, November, 2004.
- [5] B. Boehm, C. Abts, A. W. Brown, S. Chulani, B. Clark, E. Horowitz, R. Madachy, D. Reifer, and B. Steece, *Software Cost Estimation with COCOMO II*, Prentice Hall, Upper Saddle River, NJ, 2000.
- [6] B. Boehm, B. Clark, E. Horowitz, and C. Westland, *The COCOMO 2.0 Software Cost Estimation Model*, University of Southern California, 1995.
- [7] J. Herbsleb, A. Carleton, J. Rozum, J. Siegel, and D. Zubrow, *Benefits of CMM-based software process improvement: Initial results*, Technical Report CMU/SEI-94-TR-013, Software Engineering Institute, Pittsburgh, Pennsylvania, 1994.
- [8] J. D. Arthur, M. K. Groener, K. J. Hayhurst, and C. M. Holloway, Evaluating the effectiveness of independent verification and validation, *IEEE Computer*, October, 1999, 79 – 83.
- [9] G. Page, F. E. McGarry, and D. N. Card, A practical experience with independent verification and validation, *Proceedings of the 8th International Computer Software and Applications Conference*, IEEE Computer Society Press, 1984.
- [10] R. A. Rogers, D. B. McCaugherty, and F. Martin, A case study on IV&V Return on Investment, *Proceedings of the NDIA 3rd Annual Systems Engineering and Supportability Conference*, 2000.
- [11] C. Jones, *Software Quality: Analysis and Guidelines for Success*, International Thomson Computer Press, Boston, MA, 1997.
- [12] N. Eickelmann, A. Anant, J. Baik, and W. Harrison, *Developing Risk-Based Financial Analysis Tools and Techniques to Aid IV&V Decision Making*, NASA Contract S-54493-G Technical Report, NASA IV&V Facility, Fairmont, WV, 2001.
- [13] B. W. Boehm, *Software Engineering Economics*, Prentice-Hall, Englewood Cliffs, NJ, 1981.
- [14] J. Rothman, *What does it cost you to fix a defect? And why should you care?* Rothman Consulting Group, Inc., www.jrothman.com, October, 2000.
- [15] J. Rothman, *What does it cost to fix a defect?* www.stickyminds.com, February, 2002.
- [16] T. McGibbon, *A business case for software process improvement*, Data & Analysis Center for Software, Air Force Research Laboratory – Information Directorate, Rome, NY.
- [17] *Case study: Finding defects earlier yields enormous savings*, Cigital, www.cigital.com, 2003.
- [18] G. M. Schneider, J. Martin, and W. T. Tsai, An experimental study of fault detection in user requirements documents, *ACM Transactions on Software Engineering and Methodology*, 1(2), April, 1992, 188 – 204.
- [19] *From Software Quality Control to Quality Assurance*, Mortice Kern Systems Inc., 2001.
- [20] S. Pavlina, *Zero-defect software development*, Dexterity Software, www.dexterity.com, 2001.
- [21] *Parametric Estimating Handbook*, U.S. Department of Defense, 1999.
- [22] *Function Point Counting Practices Manual*, Release 4.1.1, The International Function Point Users' Group, 2000.
- [23] C. Jones, "Software defect-removal efficiency," *IEEE Computer*, Vol. 29, No. 4, 1996, pp. 94 – 95