Hindawi
Mathematical Problems in Engineering
Volume 2021, Article ID 7837153, 10 pages
https://doi.org/10.1155/2021/7837153

Research Article

Cost Benefit Analysis of Incorporating Security and Evaluation of Its Effects on Various Phases of Agile Software Development

Sushil Kumar[1], **Avinash Kaur**[1], **Ashish Jolly**[2], **Mohammed Baz**[3] and **Omar Cheikhrouhou**[4]

[1]*Department of Computer Science and Engineering, Lovely Professional University, Jalandhar, Punjab, India*
[2]*Department of Computer Science, Government PG College, Ambala Cantt, Haryana, India*
[3]*Department of Computer Engineering, College of Computer and Information Technology, Taif University, PO Box. 11099, Taif 21994, Saudi Arabia*
[4]*CES Laboratory, National School of Engineers of Sfax, University of Sfax, Sfax 3038, Tunisia*

Correspondence should be addressed to Omar Cheikhrouhou; omar.cheikhrouhou@isetsf.rnu.tn

Received 3 July 2021; Accepted 22 July 2021; Published 3 August 2021

Academic Editor: Dilbag Singh

This article addresses the costs and benefits of integrating security into the development of applications and gives formulas for calculating security costs and benefits. The lack of safe application might lead to safety issues. Increasingly, there are accidents recorded that expose security flaws in many major software systems. It results in significant losses for consumer companies. While software businesses are working to produce secure software, the utility of secure software is quite limited. In contrast to the traditional manufacturers of commodities, for example, automakers, software developers have no legal responsibility if their products include flaws. The market reacts adversely to software manufacturers with serious vulnerabilities in their products. This is because of the loss of credibility, cost of patches, and so on. The study shows that the market is ready to penalize the supplier for insecurity and therefore offers the chance to deliver safer technologies. To improve cost/efficiency, the vulnerabilities are connected by accessible fixes. Significant savings are gained when security shortcomings are corrected during designing requirements instead of fixing security failures after deploying software. For suppliers, updates are more expensive to produce and publish. In addition, development costs can be reduced by plugging security issues in the early stages of development.

1. Introduction

The major hurdles in determining the cost of building safe software are a lack of accurate data, a lack of agreement on measuring methods, and a relatively new focus on safety. In addition, considerable quantification and software development cost assessments are being undertaken. Work began with the COCOMO (Constructive Cost Model) [1]. Security costs are directly commensurated with software quality. In general, improved software quality involves better design, aggressive testing, and validation, all of which have a direct impact on costs. The advantages of such transactions may, in addition, nevertheless exceed the expenses. The costs for greater quality are divided into "compliant" and "non-compliant." Compliance means the specified quality may be achieved. To attain the proper quality, either (1) remove the origin of defects (improved training, meetings on quality improvement, and design reviews) or (2) remove flaws through product evaluation and monitoring (code inspection, screening, and software calibration activities)

The financial return assessment for improved quality is done on the basis of the data used by the company and shows that this ROI is promising and significant.

Sadly, any cost metric for software quality is a challenge because these activities are often brand, technology, or organization-specific and so impossible to identify. Rather, it has been proven that intelligent employees, the use of case tools, and the considerably more transparency in planning, design, and so on boost product quality [2, 3]. Greater quality minimizes the frequency of bugs, but it does not

influence the product size, does not affect the product's complexity, and so on. Improving the product nevertheless can entail more or less product complexity, more or less size, and more or less product flexibility. Therefore, the design of the product can incur indirect costs that exceed the cost of compliance, such as the costs of case tools [4].

To comprehend the benefits of safe software, we focus on research that explains methodologies and frameworks to support customer information security decision-making, i.e., from the viewpoint of an IT-based organization that wishes to defend itself against cyber-assault. This research is relevant as long as secure software development is another option to invest in information safety [5, 6].

We adapt this approach to the nature of software development and to evaluate investment returns for secure software development techniques. In general, the benefits are computed as the defined cash worth of losses saved by safe production methods of software. The principal discrepancies in the results are how the averted losses are measured. The data required to evaluate damage (including potentially safe software development strategies) with or without security expenses were based on operational data. Bypass rate calculation here is a crucial issue. Current security methods prohibit bypass rates from estimating the fraction of failure (for example, invasion of privacy, assault, virus infections, and internal theft) and thus not recorded in administrative data on detected intrusions or losses. To determine the average rate of that sort of loss event, the identified rate of a particular loss incident is multiplied by the inversion of the bypass rate. The amount of damage experienced by a company in relation to the loss case, computed in dollars, is measured by administrative statistics or other means (such as assessments by managers) independently [4].

The manner in which loss data are produced is another important difference. This literature is based on the approach to the interpretation of the importance of the market by recognizing that the major advance of safe software development is that software users may have a greater risk avoidance compared with baseline [7]. Therefore, the value of secure software includes parts equivalent to that of investment in the protection of information.

The collection of the data will be tough. In general, although the developer is a member of an organization, it is unclear if the developer would be able to make an assessment of the frequency and related losses of failures in administrative records [8, 9]. Developers too cannot estimate these amounts. In addition, developers will not be able to evaluate the proportion of the benefits they have generated. Vendors are likely to be more concerned about the loss of credibility and a loss of potential revenue.

Other prospective advantages are also available. It is also important because fewer software bugs will require fewer fixes and improvements in security. As a result, a second component of the benefit of safe software development is the decrease in patching expenses. The risk mechanism for this may be altered while there is no established technique for determining the averted patching cost. The gain should therefore be estimated as the difference between the basic patching cost and the anticipated patching cost [10]. The basic cost of patches is calculated by the reported number of vulnerabilities per thousand lines of code per year multiplied by the detection rate, multiplied by the average vulnerability cost of patches. The discovery rate is needed in this scenario because not all vulnerabilities in the system are likely to be identified.

Consequently, our suggested method is versatile, allowing users to quantify the value in other ways. However, we are careful to reduce the chance of duplication.

2. Calculating Security Benefits

Most companies are now going towards agile applications. The financial impact of Agile encourages other businesses to move to agile practices. CIOs today are also highly concerned about the financial analysis of Agile vs. Waterfall. The important elements that cause many IT companies to get rid of the traditional waterfall process are costs, benefits, ROI, NPV, and ROA [11]. Some statistics from a recognized study are presented below to highlight the advantages of agile software development over conventional software development [12, 13].

2.1. Costs. The project needs to be accomplished by an organization within the budget. As the recent product introduction becomes faster than ever, projects under budgetary limitations are increasingly vital.

2.2. Benefits. The agile is more effective because there are fewer defects in the end output. As the quality of the product becomes good with several revisions, the product is tested for a number of times due to iterations that lead to greater product quality.

2.3. ROI. Agile offers more benefits than the traditional strategy at a low cost. The higher profit-cost differential makes ROIs (return on capital) approximately seven times higher than the traditional solution possible.

2.4. NPV. NPV is the difference between cash influx and cash outflow for a period of time. This is about 200% higher than traditional NPV.

2.5. ROA. It offers us an indication of the properties being used efficiently. The assets are only utilized for the project in the typical way. This reduces the idle period of the capital of a project. However, the items are utilized at the optimum level due to their adaptable character. The increase in idle time would put the project at risk. ROA will therefore also allow us to understand the hazards of any undertaking. Agile reduces the project load by almost 140% compared with conventional techniques.

Figure 1 shows the security benefit analysis in terms of traditional and agile software development.

To estimate the advantages of using security for a software project [14], the following details must be collected:

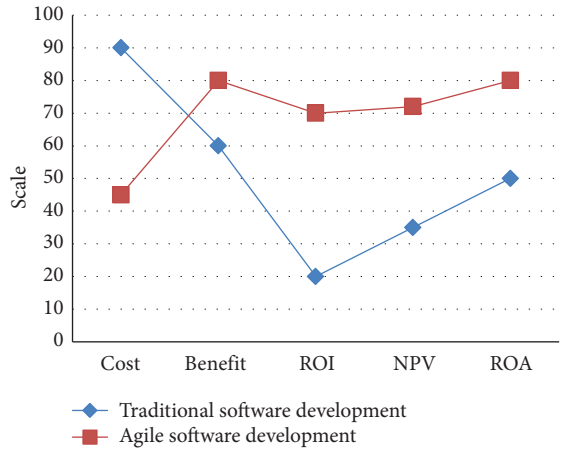

Figure 1: Security benefit analysis in terms of traditional and agile software development.

(1) *Size of Software*. It is the number of source code lines.

(2) *Bug Rate*. The number of bugs per thousand source code lines occurred in the software (tsloc) (both for security and nonsecurity bugs).

(3) *Costs for Errors*. It is the average error cost. The bug costs are separated into the costs of prerelease and postrelease [15].

(4) *Prerelease Component*. It is the rate at which flaws are detected and addressed prior to release, or the average cost of bug fixing prior to software release.

(5) *Postrelease Component*. It is the percentage of security bugs that attackers believe to discover and use [18].

(5a) Overheads for media relations, including man-month and any additional costs.

(5b) Legal fees, including man-month expenditures and any additional costs incurred.

(5c) Man-month cost of customer service—the effect on future income lost because of security infringement [16]. Revenues are anticipated to return within one year.

(5d) Extraincidental expenses in dollars.

(5e) Extraincidental expenses in dollars—Total cost of postrelease diagnosis and incidental costs.

(5f) The whole postrelease patching costs for humans including accidentals.

(5g) The overall expense for months of software postrelease testing and incidentals.

(5h) The average cost of each user per person per month.

(6) *Postsecurity Component*. Bug detection in prerelease increases on average when protection and checks are enhanced. The advantage is measured by two alternative calculations as follows:

(6a) The approximate decrease (by age) in the overall number of defects caused by enhanced security standards and procedures [17] is measured. One evaluates the projected losses prior to adding security, whereas the other calculates the anticipated expenditures after adding security.

The prerelease components can be computed as follows:

$$C_{[\mathrm{pr}]} = B_{[\mathrm{per}\,d]} * F_{[\mathrm{cpr}]}. \tag{1}$$

Here, $C_{[\mathrm{pr}]}$ defines the prerelease component; $B_{[\mathrm{per}\,d]}$ represents the percentage of bugs discovered; and $F_{[\mathrm{cpr}]}$ shows the fixed cost during prerelease.

$$C_{[\mathrm{e}]} = \left(\left(\left(\frac{N_{[\mathrm{sb}]}}{\left(N_{[\mathrm{sb}]} + \left(N_{[\mathrm{nsb}]}\right)\right)}\right) * \left(1 - B_{[\mathrm{pd}]}\right)\right) * \left(B_{[\mathrm{PC}]} * T_{[c]}\right)\right). \tag{2}$$

Here, $C_{[\mathrm{e}]}$ defines the exploit component; $N_{[\mathrm{sb}]}$ represents the number of securities [18] with bugs; $N_{[\mathrm{nsb}]}$ is the number of nonsecurity bugs; $B_{[\mathrm{pd}]}$ is the percentage of bugs discovered; $B_{[\mathrm{pe}]}$ represents the percentage of bugs exploited; and $T_{[c]}$ is the total cost and can be computed as follows:

$$T_{[c]} = T_{[\mathrm{prc}]} + T_{[\mathrm{lc}]} + T_{[\mathrm{csc}]} + P_{[l]} + C_{[o]}, \tag{3}$$

where $T_{[\mathrm{prc}]}$ represents the total public relation cost; $T_{[\mathrm{lc}]}$ shows the total legal cost; $T_{[\mathrm{csc}]}$ shows the total client support cost; $P_{[l]}$ represents the profit lost; and $C_{[o]}$ shows the other costs as follows:

$$P_{[l]} = \left(S_{[\mathrm{pl}]} * R_{[\mathrm{ts}]}\right) * \left(P_{[m]}\right). \tag{4}$$

Here, $S_{[pl]}$ shows the percentage of sales lost; $R_{[ts]}$ represents the revenue from total sales; and $P_{[m]}$ depicts the profit margin. Component postrelease can be computed as follows:

$$C_{[por]} = \left(1 - D_{[ppor]}\right) * T_{[cpor]}. \quad (5)$$

Here, $C_{[por]}$ shows the postrelease component; $D_{[ppor]}$ represents the percent discovered postrelease; and $T_{[cpor]}$ defines the total cost postrelease.

$$T_{[cpor]} = T_{[dc]} + T_{[pc]} + T_{[ttc]}. \quad (6)$$

Here, $T_{[dc]}$ shows the total diagnostic cost; $T_{[pc]}$ represents the total patch cost; and $T_{[tc]}$ defines the total testing cost. Expected cost presecurity [19] can be computed as follows:

$$C_{[eps]} = \left(C_{[pr]} + C_{[e]} + C_{[por]}\right) * B_{[n]} * S_{[p]}. \quad (7)$$

Here, $C_{[eps]}$ defines the expected cost presecurity; $C_{[pr]}$ shows the component prerelease; $C_{[e]}$ represents the component exploit; $C_{[por]}$ defines the component postrelease; $B_{[n]}$ represents the number of bugs; $S_{[p]}$ defines the size of project; $B_{[n]}$ can be computed as $B_{[s]} + B_{[ns]}$ where $B_{[s]}$ represents the security bugs and $B_{[ns]}$ denotes the nonsecurity bug; and $S_{[p]}$ represents the size of the project and can be computed using $S_{[p]} = N_{[loc]}/1000$ where $N_{[loc]}$ defines the no of the lines of code. Prerelease constituents can be computed as follows:

$$C_{[pr]} = \left(B_{[pd]} * \left(1 + B_{[ipd]}\right) * C_{[prf]}\right). \quad (8)$$

Here, $C_{[pr]}$ shows the prerelease constituent; $B_{[pd]}$ defines the percentage of bugs discovered prerelease; $B_{[ipd]}$ shows the percentage of increase in bug prereleases; and $C_{[prf]}$ shows the prerelease fix cost. Exploit constituents can be computed as follows:

$$C_{[e]} = \left(\frac{N_{[sb]}}{\left(N_{[sb]} + N_{[nsb]}\right)}\right) * \left(1 - B_{[pd]}\right) * \left(B_{[pe]} * T_{[c]}\right). \quad (9)$$

Here, $C_{[e]}$ shows the exploit constituent; $N_{[sb]}$ defines the number of security bugs; $N_{[nsb]}$ demonstrates the number of nonsecurity bugs; $B_{[pd]}$ shows the percentages of bugs discovered prerelease; $B_{[pe]}$ shows the percentages of bugs exploited; and $T_{[c]}$ shows the total cost and can be computed as $T_{[c]} = T_{[prc]} + T_{[lc]} + T_{[csc]} + P_{[l]} + C_{[o]}$, where $T_{[prc]}$ shows the total public relation cost, $T_{[lc]}$ represents the total legal cost, $T_{[csc]}$ demonstrates the total client support cost, $P_{[l]}$ depicts the profit lost, $C_{[o]}$ represents the other costs, and $P_{[l]}$ can be computed as $(S_{[pl]} * R_{[ts]}) * (P_{[m]})$, in which $S_{[pl]}$ shows the percentages of sales lost, $R_{[ts]}$ shows the revenue from total sales, and $P_{[m]}$ shows the profit margin. It can be computed as follows:

$$C_{[epos]} = \left(C_{[pr]} + C_{[e]} + C_{[por]}\right) * B_{[n]} * S_{[p]}. \quad (10)$$

Here, $C_{[epos]}$ shows the expected cost postsecurity; $C_{[pr]}$ demonstrates the prerelease component; $C_{[e]}$ defines the component exploit; $C_{[por]}$ shows the postrelease component; $B_{[n]}$ is the number of bugs; and $S_{[p]}$ defines the project size. $B_{[n]} = B_{[s]} + B_{[ns]}$, where $B_{[s]}$ shows the security bugs and $B_{[ns]}$ denotes the nonsecurity bug. $S_{[p]} = N_{[loc]}/1000$, where $S_{[p]}$ shows the size of project and $N_{[loc]}$ represents the number of lines of code.

Figure 2 shows the value of various cost drivers with reference to security incorporated prerelease or postrelease. It distinguishes between the security prerelease and security postrelease.

3. Calculation of Security Costs

To estimate the cost of secure development, we modify the current models to include security aspects. Recent work in the development of secure software tries to embed security features in proven cost estimating models [20, 21] (particularly COCOMO-II). The main idea behind this method is that enhancing safety is likely to increase the effort necessary to make the product. Most conceptually, as $E_\alpha - E = E_\beta$, where E signifies the amount of effort in the person month and ΔE, supplementary effort is needed to create a secure item, E_α is the effort required with security, and E_β is the effort required without security. E_α is evaluated with a certain certainty even though COCOMO-II is widely used in computing E_β and customers are accustomed to these models. The effort level E formula (in person months) is provided as E (estimated) = aKLOCSF x Π (EM), where KLOC denotes the number of 1000 code lines and SF denotes the scaling factor. SF = 1.01 + 0.01 SUM (WI), where 5 scaling factors are used for WI. EM is a multiplier of effort. Every WI and EM is classified at a very low, nominal, moderate, and very high level. The sensitivity of each factor is calculated and tends to develop on the basis of the particular project setup.

3.1. Major Items of Cost.

(i) Using innovative CASE tools for the development of secure applications.

(ii) *User Training.* Security requirements [22] will allow the company to provide the developers with more training. The cost will increase because of the following:

(a) The cost of direct training, i.e., recruitment and payment for the training of employees by others.

(b) An opportunity cost in terms of time while employees are attending training sessions. They all need to be combined.

(iii) Raising the amount of effort due to added safety, thereby increasing the cost.

(iv) The impact of product delivery delay is due to increased surveillance.

More work could lead to a project delay; the details required from the right project personnel are as follows:

(a) The percentage difference is determined in code size by adding encryption [23, 24]. Team members, for example, will expect that improved

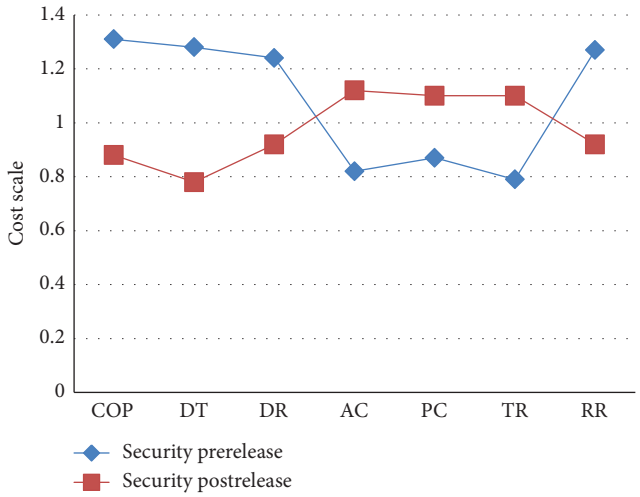

Figure 2: Cost drivers with reference to security incorporated prerelease or postrelease.

security protocols would increase the size of the device by 5% in terms of the number of lines of code.

(b) The approximation of the software project's complexity (from very low to very high) before and after encryption is implemented. For example, a team member may believe that the complexity of the software project was low prior to the implementation of security protocols but then increased to moderate or extreme.

(c) The necessary amount of program documents before and after security measures is predicted.

(d) System analyst ability is approximated prior to and after safety implementation.

(e) An expected ability of the programming team before and after security is implemented.

(f) A rough estimate of experience on the tools needed to add protection to projects.

(g) The anticipated change in production time prior to and following the implementation of safety measures is estimated.

(h) An approximation of the overall commitment (in men's months) is mandatory to construct the software prior to implementing protection.

(i) The total expense per worker per month is predicted, which is the amount charged for 30 days of working time on average.

(j) The reliability parameters prior to and after protection are assessed. Customer-required training costs are estimated. The parameters considered for calculating training costs are total on-job employees, time spent on training by an individual employee, and per day, the cost of training.

An estimate of the losses incurred as a result of delayed business entry is depicted by the following equation:

$$C_{[e]} = E_{[n]} * C_{[\text{ppm}]}. \tag{11}$$

Here, $E_{[n]} = E_{[o]} + E_{[c]}$; $C_{[e]}$ shows the effort cost; $E_{[n]}$ depicts the new effort; $C_{[\text{ppm}]}$ defines the cost per person month; $E_{[o]}$ defines the effort old; and $E_{[c]}$ shows the effort change. The effort change can be computed as follows:

$$E_{[c]} = \left(\left(1 + C_{[\text{psi}]}\right) \wedge 1.15\right) * \left(\frac{C_{[b]}}{C_{[a]}}\right) * \left(\frac{D_{[b]}}{D_{[a]}}\right) * \left(\frac{\text{AC}_{[b]}}{\text{AC}_{[a]}}\right) * \left(\frac{\text{PC}_{[b]}}{\text{PC}_{[a]}}\right) * \left(\frac{\text{TO}_{[b]}}{\text{TO}_{[a]}}\right) * \left(\frac{T_{[b]}}{T_{[a]}}\right) * \left(\frac{R_{[b]}}{R_{[a]}}\right). \tag{12}$$

Here, $E_{[c]}$ shows the effort change; $C_{[\text{psi}]}$ defines the percentage code size increase; $C_{[b]}$ shows the complexity before; $C_{[a]}$ represents the complexity; $D_{[b]}$ shows the documentation before; $D_{[a]}$ defines the documentation; $\text{AC}_{[b]}$ represents the analyst capability before; $\text{AC}_{[a]}$ defines the analyst capability; $\text{PC}_{[b]}$ shows the programmer capability before; $\text{PC}_{[a]}$ represents the programmer capability; $\text{TO}_{[b]}$ defines the tools before; $\text{TO}_{[a]}$ represents the tools; $T_{[b]}$ defines the time before; $T_{[a]}$ represents the time; $R_{[b]}$ defines the reliability before; and $R_{[a]}$ shows the reliability.

The numerator and denominator for each of the words in the "Effort-Change" equation are taken from the developer's responses to the respective standards above and assigned a numeric value accordingly. We have selected COCOMO because it is well established and nonproprietary [25].

Figure 3 depicts the approximate complexity of the project before and after the security is integrated. It distinguishes between the complexity of presecurity and the complexity of the project postsecurity cost scale.

Figure 4 shows the estimation of the expected shift in development time before and after the introduction of security processes. It distinguishes between the development time presecurity and development time postsecurity.

Figure 5 shows the approximation of the amount of documentation needed beforehand and afterward the integration of security. It distinguishes between the documentation required presecurity and documentation required postsecurity.

Figure 6 depicts the approximate system analyst capability when presecurity and postsecurity are implemented. It compares between the analyst capability presecurity and analyst capability postsecurity values.

Figure 7 shows the approximate programmer capability when presecurity and postsecurity are implemented. It compares between the programmer capability presecurity and programmer capability postsecurity cost scales.

Figure 8 depicts an estimate of the resources needed to add security to the given project. The experience of team members with these resources presecurity and postsecurity deployment was taken into consideration.

Figure 9 shows an estimation of the reliability specifications presecurity and postsecurity incorporation.

Figure 10 demonstrates the corresponding value of each cost driver for each option (very low to very high) whichever is appropriate. The cost scale is very high, high, moderate,

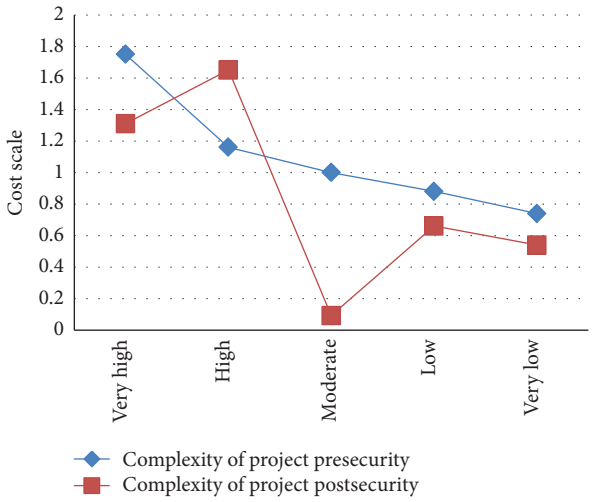

Figure 3: Complexity of the project with reference to cost scale.

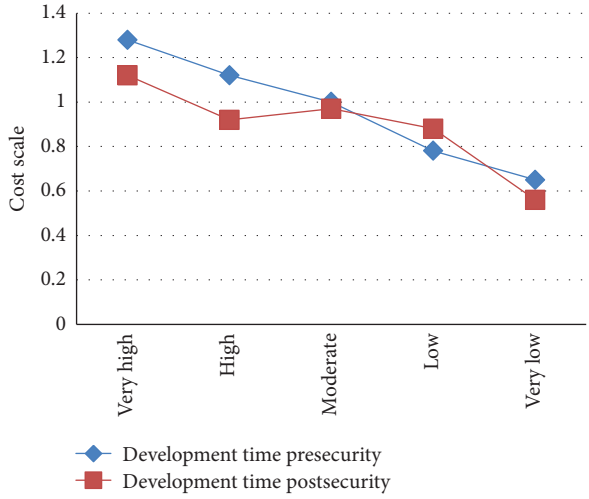

Figure 4: Development time of project with reference to cost scale.

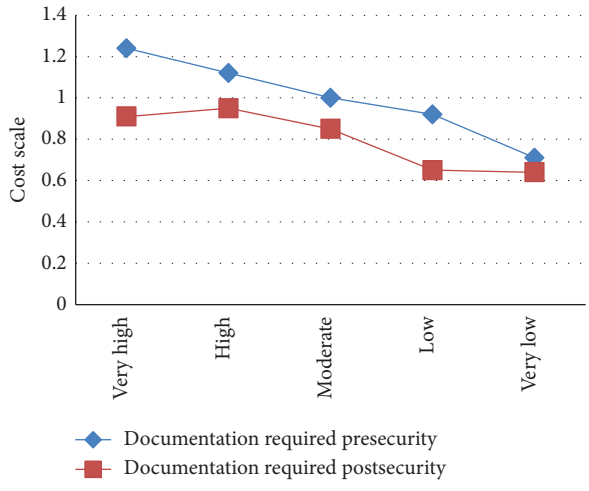

Figure 5: Documentation required in software development with reference to cost scale.

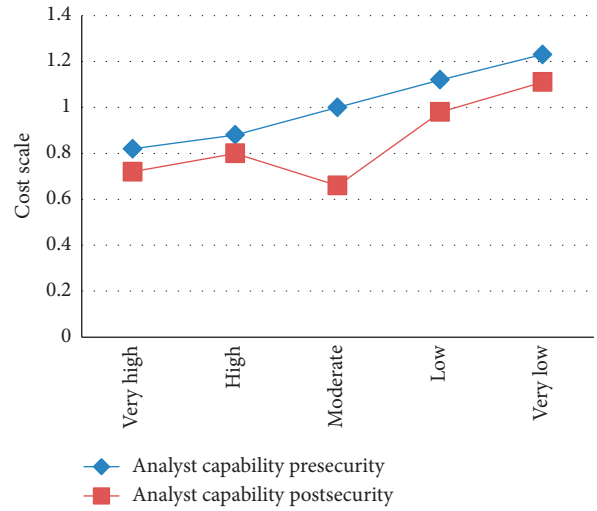

Figure 6: Analyst capability in software development with reference to cost scale.

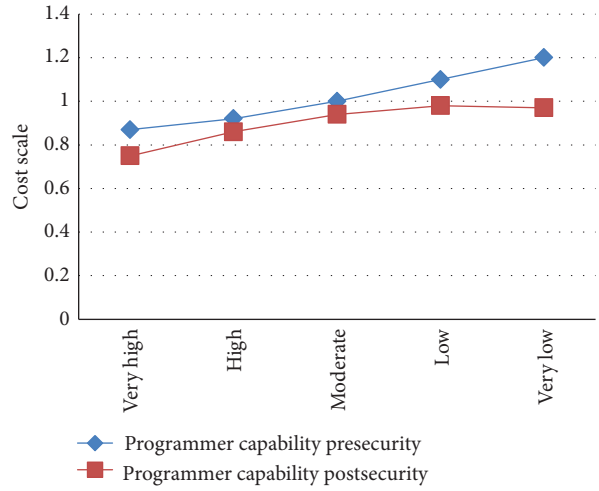

Figure 7: Programmer capability in software development.

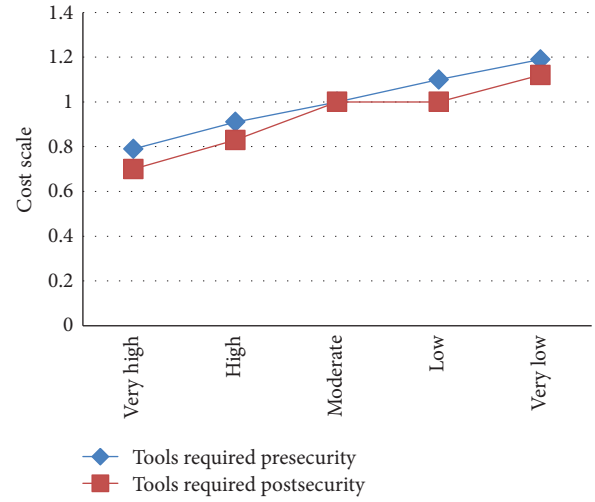

Figure 8: Tools required in software development.

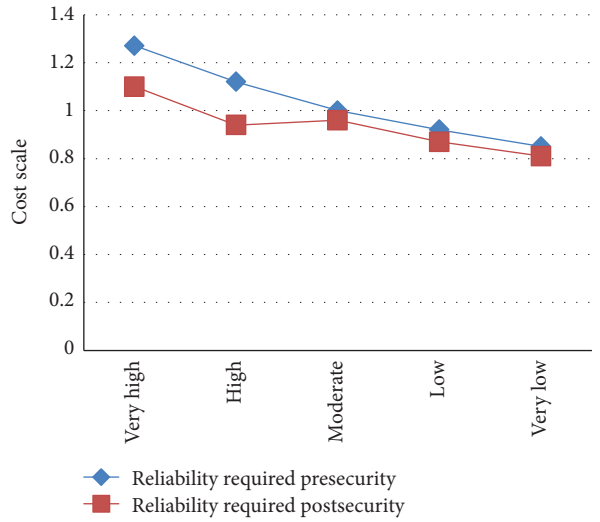

Figure 9: Impact of reliability on the cost of software development.

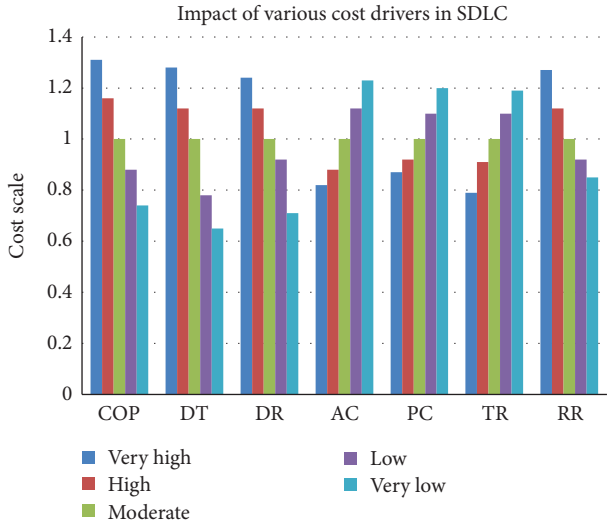

Figure 10: Visualization of the corresponding value of each cost driver and cost scale.

low, and very low, whereas the cost drivers are COP (complexity of project), DT (development time), DR (documentation required), AC (analyst capability), PC (programmer capability), TR (tools required), and RR (required reliability).

Figure 11 shows the correlation between cost and various components affecting it, like the number of bugs which will increase in case of security incorporated prerelease because of improved protection and control; subsequently, the size of the project, the effort required, and development time will be more as security is incorporated prerelease, thereby increasing the overall cost. The maintenance of the project will be less in case the security incorporated [26] prerelease, thereby the maintenance cost will be less compared with postrelease incorporation of security. The planning and quality of the final product will be better in case of prerelease incorporation of security. Opportunity costs can be computed as follows:

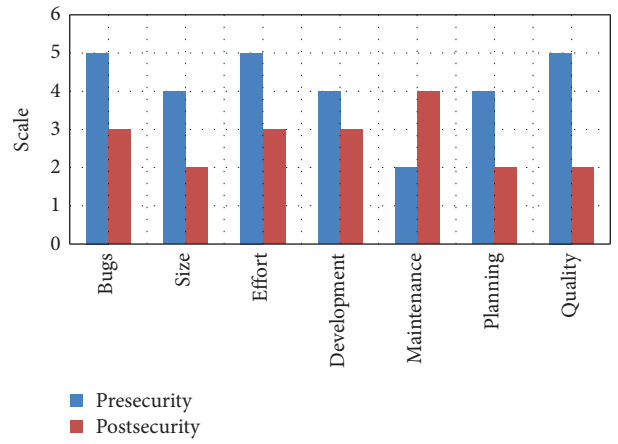

Figure 11: Correlation of various components affecting the cost of software development.

$$C_{[o]} = E_{[\mathrm{it}]} * T_{[l]} * \left(\frac{C_{[\mathrm{aec}]}}{365}\right). \tag{13}$$

Here, $C_{[o]}$ represents the opportunity cost; $E_{[\mathrm{it}]}$ defines the number of employees in training; $T_{[l]}$Is the average length of training; and $C_{[\mathrm{atc}]}$ is the average training cost. The cost of tools can be computed as follows:

$$C_{[\mathrm{to}]} = E_{[c]}. \tag{14}$$

Here, $C_{[\mathrm{to}]}$ defines the cost of case tools/hardware or software; $E_{[c]}$ shows the capital expenses for purchasing hardware or software for a project. The total cost can be computed as follows:

$$C_{[t]} = C_{[e]} + C_{[\mathrm{tr}]} + C_{[o]} + C_{[\mathrm{to}]} + C_{[d]}. \tag{15}$$

Here, $C_{[t]}$, $C_{[e]}$, $C_{[\mathrm{tr}]}$, $C_{[o]}$, $C_{[\mathrm{to}]}$, and $C_{[d]}$ represent the total cost, cost of effort, cost of training, cost of opportunity, cost of tools, and cost of delay to market.

Figure 12 shows that risk analysis is the best security activity, coding rule is the worst, and threat modeling is the average security activity to be incorporated during the planning phase.

Figure 13 shows that coding rule is the best security activity, role matrix is the worst, and vulnerability testing is the average security activity to be incorporated during the coding phase.

Figure 14 shows that security testing is the best security activity, operational planning is the worst, and identifying trust boundaries is the average security activity to be incorporated during the testing phase.

Figure 15 shows the best, worst, and average security activities during the planning, coding, and testing phases of software development. Table 1 demonstrates the best, worst, and average security activities for the planning, coding, and testing phases of software development. Figure 15 and Table 1 clearly indicate the various risks associated during the various phases of software development.

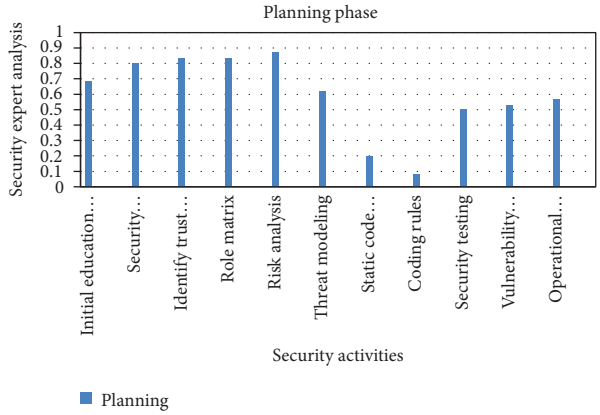

Figure 12: Best, worst, and average security activities in the planning phase.

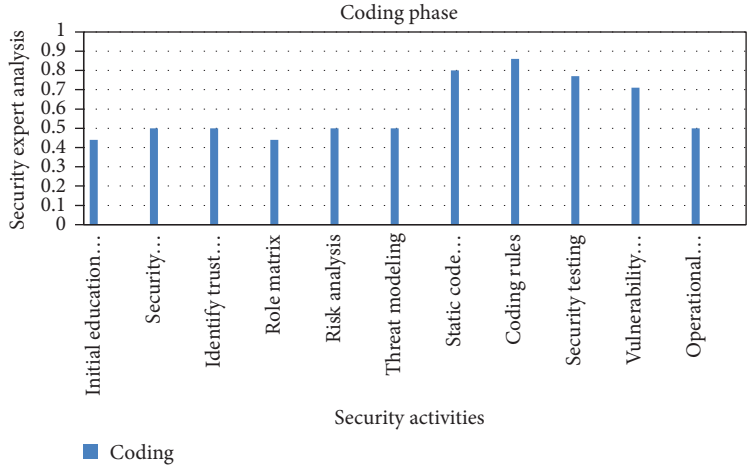

Figure 13: Best, worst, and average security activity in the coding phase.

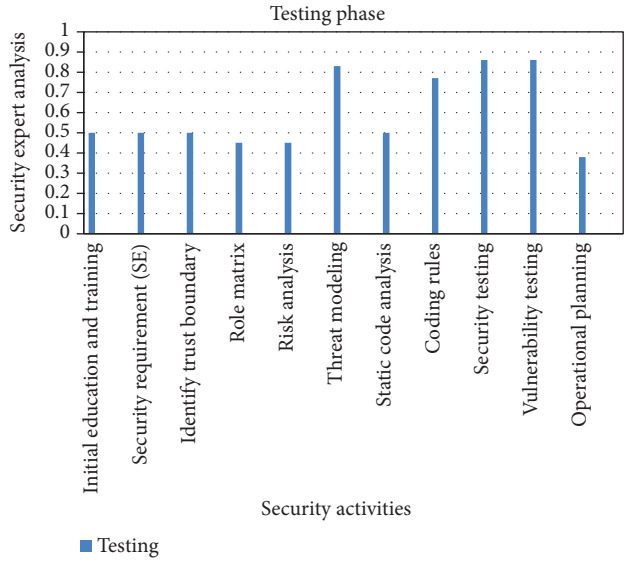

Figure 14: Best, worst, and average security activity in the testing phase.

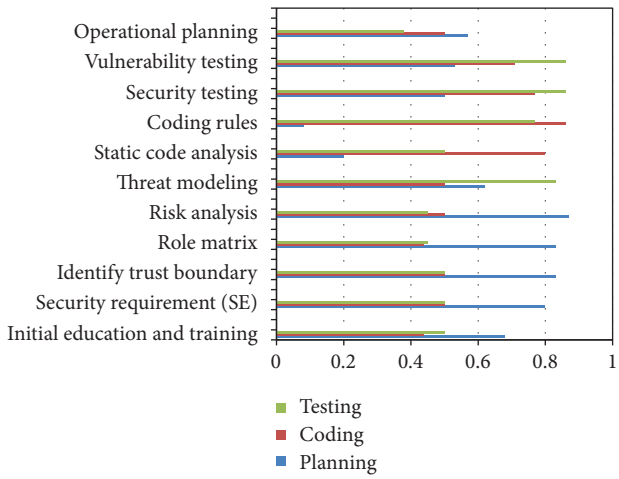

Figure 15: Best, worst, and average security activities during the planning, coding, and testing phases of software development.

Table 1: The best, worst, and average security activities for the planning, coding, and testing phases of software development.

	Best security activity	Worst security activity	Average security activity
Planning phase	Risk analysis	Coding rule	Threat modeling
Coding phase	Coding rules	Role matrix	Vulnerability testing
Testing phase	Security testing	Operational planning	Identify trust boundaries

4. Conclusion

The cost and benefit of each project will differ. Furthermore, cost and benefit can be divided into two categories: tangible and intangible cost and benefit. Hardware prices, professional wages, and software expenditures are all examples of tangible costs. They are measured and evaluated. Examples of tangible costs include the acquisition of hardware or software, employee training, and staff compensation. Intangible costs are those that cannot be quantified. The expense of a malfunction of an online system during banking hours will result in the bank losing money. Benefits are also tangible or intangible. For example, more customer satisfaction, improved company status, and so on are all intangible benefits whereas improved response time and producing error free output such as producing reports are all tangible benefits. Both tangible and intangible costs and benefits should be considered in the evaluation process.

This article describes the economic advantage of adding security during several agile software processes. The examination of the cost benefits is divided in two parts: the prerelease portion and the postrelease section. The general bug identification rate is increasing in prerelease due to improved security and control. Two distinct methods are used to calculate the benefit. The total number of bugs was projected to reduce due to the higher safety standards and controls. First, the estimated losses are computed before the security is added, and second the expected costs are calculated after the security is added. We should presume that the benefits of security integration outweigh the risks. It has been discovered that addressing security concerns during the initial phase costs up to 100x times less than addressing flaws in the released software. The article also defines each person's role, from the top of management down the hierarchy, in promoting the model and establishing a sustainable application [27–30].

Data Availability

The data that support the findings of this study are available upon request to the corresponding author.

Conflicts of Interest

The authors declare that they have no conflicts of interest.

Acknowledgments

The authors thank Taif University Research Supporting Project (TURSP-2020/239), Taif University, Taif, Saudi Arabia.

References

[1] M. Siponen, R. Baskerville, and T. Kuivalainen, "Integrating security into agile development methods," in *Proceedings of the 38th Annual Hawaii International*, Big Island, Hawaii, January 2005.

[2] S. Sonia and S. Archana, "Integration analysis of security activities from the perspective of agility," *IEEE*, vol. 9, pp. 40–47, 2012.

[3] A. Arora, D. Hall, C. A. Piato, D. Ramsey, and R. Telang, "Measuring the risk-based value of IT security solutions," *IEEE IT Professional*, vol. 6, no. 6, pp. 35–42, 2004.

[4] M. S. Krishnan, C. M. Howard, and S. Lipner, *The Security Development Lifecycle – SDL: A Process for Developing*

Demonstrably More Secure Software, Microsoft Press, Washington, DC, USA, 2006.

[5] B. Boehm, J. R. Brown, and J Lipow, "Quantitative evaluation of software quality," in *Proceedings of the 20th International Conference on Software Engineering*, Washington, DC, USA, May 1976.

[6] B. Boehm, E. Colbert, Y. Chen, D. Wu, and D. Reifer, *Costing the Development of Secure Systems Eighth Annual PSM Users' Group Conference: Measurement for Enterprise Excellence*, Keystone, CO, USA, 2004.

[7] H. Cavusoglu, B. Mishra, and S. Raghunathan, "The effect of internet security breach announcements on market value: capital market reactions for breached firms and internet security developers," *International Journal of Electronic Commerce*, vol. 9, pp. 69–104, 2004.

[8] S. Butler, *Security Attribute Evaluation Method: A Cost-Benefit Approach*, IEEE, Orlando, FL, USA, 2002.

[9] K. Campbell, L. A. Gordon, M. P. Loeb, and L. Zhou, "The economic cost of pub-licly announced information security breaches: empirical evidence from the stock market," *Journal of Computer Security*, vol. 11, no. 3, pp. 431–448, 2003.

[10] K. B. Hendricks and V. R. Singhal, "Delays in new product introductions and the market value of the firm: the consequences of being late to the market," *Management Science*, vol. 43, no. 4, pp. 422–436, 1997.

[11] C. Kemerer, "An empirical validation of software cost estimation models May 1987, 416 quality and user satisfaction of software products: an em-pirical analysis," ACM, Technical Report, 1997.

[12] M. S. Krishnan, C. H. Kriebel, S. Kekre, and T. Mukhopadhyay, "An empirical analysis of productivity and quality in software products," *Management Science*, vol. 46, no. 6, pp. 745–759, 2000.

[13] MSDN, "Microsoft security development lifecycle (SDL)," *Measurable Im-Provements for Flagship Microsoft Products*, vol. 99, 2008.

[14] E. Rescorla, *Is Finding Security Holes a Good Idea? the Third Workshop on Eco-Nomics and Information Security*, IEEE, Minneapolis, MN, USA, 2004.

[15] M. Zulkernine and I. A. Sheikh, *Software Security Engineering: Towards Unifying Software Engineering and Security Engineering*, Queen's University, Kingston, Canada, 2009.

[16] W. Wang, K. Reddy Mahakala, A. Gupta, and N. Hussein, *Data on Security Requirements in Open-Source Software Projects, Data in Brief*, Elsevier, Amsterdam, Netherlands, 2018.

[17] W. Wang, K. Reddy Mahakalaa, A. Guptaa, and N. Husseina, *A Linear Classifier Based Approach for Identifying Security Requirements in Open Source Software Development Journal of Industrial Information Integration*, Elsevier, New York NY, USA, 2018.

[18] A. Mohammad, Ja'far Alqatawna, and M. Abushariah, "Secure software engineering: evaluation of emerging trends," in *Proceedings of the International Conference on Information Technology*, pp. 814–818, IEEE, Hyderabad, India, April 2017.

[19] M. Kaur, D. Singh, V. Kumar, B. B. Gupta, and A. A. A. El-Latif, "Secure and energy efficient based E-health care framework for green internet of things," *IEEE Transactions on Green Communications and Networking*, vol. 99, p. 1, 2021.

[20] Z. Ádám Mann, *Secure Software Placement and Configuration Future Generation Computer Systems*, Elsevier, Amsterdam, Netherlands, 2020.

[21] S. Hosseinzadeh, S. Laurén, J.-M. Holvitie, J. M. Mäkelän, S. Hyrynsalmi, and V. Leppänen, "Diversification and obfuscation techniques for software security: a systematic literature review," *Information and Software Technology*, vol. 104, pp. 72–93, 2018.

[22] P. Liu and Su Jin, *Research on Software Security Vulnerability Detection Technology International Conference on Computer Science and Network Technology*, IEEE, Harbin, China, 2011.

[23] M. Kaur and D. Singh, "Multiobjective evolutionary optimization techniques based hyperchaotic map and their applications in image encryption," *Multidimensional Systems and Signal Processing*, vol. 32, no. 1, pp. 281–301, 2021.

[24] M. Kaur, D. Singh, and V. Kumar, "Color image encryption using minimax differential evolution-based 7D hyper-chaotic map," *Applied Physics B*, vol. 126, no. 9, p. 147, 2020.

[25] M. Saito, A. Hazeyama, N. Yoshioka et al., "A case-based management system for secure software development using software security knowledge," *Procedia Computer Science*, vol. 60, pp. 1092–1100, 2015.

[26] K. Tuma, G. Calikli, and R. Scandariato, "Threat analysis of software systems: a systematic literature review," *Journal of Systems and Software*, vol. 144, pp. 275–294, 2018.

[27] C. A. Pinto, A. Arora, D. Hall, and E. Schmitz, "Challenges to sustainable risk management," 2004.

[28] K. Soo Hoo, A. W. Sadbury, and A. R. Jaquith, "Return on security investments," *Secure Business Quarterly*, vol. 1, p. 2, 2001.

[29] N. M. Mohammeda, N. Mahmood, and M. Alshayeba, *Exploring Software Security Approaches in Software Development Lifecycle: A Systematic Mapping Study, Computer Standards and Interfaces*, Elsevier, New York NY, USA, 2016.

[30] K. Rindell, J. Ruohonen, H. Johannes, and S. Hyrynsalmi, *Security in Agile Software Development: A Practitioner Survey Information and Software Technology*, Elsevier, New York NY, USA, 2020.

[31] M. Kaur, D. Singh, and V. Kumar, "Color image encryption using minimax differential evolution-based 7D hyper-chaotic map," *Applied Physics B*, vol. 126, no. 9, p. 147, 2020.