The Journal of Systems and Software 207 (2024) 111874

Contents lists available at ScienceDirect

The Journal of Systems & Software

journal homepage: www.elsevier.com/locate/jss

The effects of required security on software development effort ☆

Elaine Venson [a,*], Bradford Clark [b], Barry Boehm [a]

[a] Center for Systems and Software Engineering, University of Southern California, Department and Organization, 941, Bloom Walk, Los Angeles, 90089, CA, USA
[b] Software Metrics Inc., Haymarket, VA, USA

ARTICLE INFO

Keywords:
Secure software development
Statistical cost model
Software security practices

ABSTRACT

The software industry has been under pressure to adopt security practices and reduce software vulnerabilities. But despite recognizing the importance of secure software development, such practices have not been broadly embraced. Costs are frequently pointed out as a barrier, although studies show that there is a lack of knowledge about the amount of resources needed to achieve a determined level of security assurance. This study quantifies the effort required to develop secure software in increasing levels of rigor and scope. We first developed an ordinal scale to quantify the degree of application of security practices. Next, we built a statistical cost model based on a data set with 1140 maintenance projects from two large companies. The model calibration revealed that the application of software security practices can impact the cost estimations ranging from 19% additional effort, on the first level of the scale, to 102% additional effort, on the highest level of the scale. These results suggest that the effort required to develop secure software is lower than it was estimated in previous studies, especially when considering the domain of Information Systems. This research builds on previous works on cost models for secure software development and goes one step further by providing empirical validation. The resulting model can be used by practitioners to estimate proper resources for adopting security practices. Additionally, the validated cost multipliers are an important piece of information for the research on secure software development investment models.

1. Introduction

Software is increasingly pervading our daily lives, bringing benefits in all areas of human activities. Nevertheless, as we become more dependent on it, concerns about the harmful impact of systems' flaws and misuse are raised, specially when they are security-related. News on data breaches and other security issues have appeared constantly on the media, and computer security has become an area of interest in industry and academia (McGraw, 2013; Amoroso, 2018; Abu-Taieh, 2017; Kott et al., 2014; Georg Schmitt, 2018).

As evidence shows that many vulnerabilities originate during software development (Hein and Saiedian, 2009; McGraw, 2006; Heitzenrater et al., 2016), organizations have published collections of recommended security practices to be applied in the software development processes (Morrison et al., 2017; Howard and Lipner, 2006; SAFECode, 2018; OWASP SAMM Project, 2017). The "building security in" mentality of Software Security, which is the idea of developing software that continues to work correctly under malicious attack, has been evolving as one response to the security problems (McGraw, 2006; Heitzenrater and Simpson, 2016c).

The Software Security approach consists of incorporating security practices during the software development life cycle (SDLC), distinguishing itself from the traditional approach to security which is focused on protecting software applications by building security around it with network-based security tools. An emerging community of Secure Software Engineering (SSE) researchers advocates for the improvement of software development processes and activities in order to deliver products that are less vulnerable to security problems and easier to protect (Heitzenrater et al., 2016; Williams et al., 2018).

Inspired by advances on Information Security Economics (ISE), the SSE community has studied the cost-effectiveness of applying security practices since the beginning of software development projects to prevent vulnerabilities (Hein and Saiedian, 2009; Bedi et al., 2013; Heitzenrater and Simpson, 2016a; Olama and Nutaro, 2013; Chehrazi et al., 2016; Heitzenrater and Simpson, 2016c; Peeters and Dyson, 2007). By identifying vulnerabilities early in the SDLC, secure software development can reduce losses from security incidents, decrease system's downtime, and save operational effort/costs (Hein and Saiedian, 2009).

☆ Editor: Alexander Serebrenik.

* Correspondence to: SQB 2 - Bloco I, Brasília, 71009-055, Brazil.
E-mail addresses: venson@usc.edu (E. Venson), brad@software-metrics.com (B. Clark), boehm@usc.edu (B. Boehm).

https://doi.org/10.1016/j.jss.2023.111874
Received 25 May 2022; Received in revised form 4 October 2023; Accepted 6 October 2023
Available online 18 October 2023

Despite the urgency to reduce vulnerabilities and the efforts from several organizations in providing developers with guidance for applying security practices, Secure Software Development Life Cycles (S-SDLC) have not been broadly embraced by the industry (Chiesa and De Luca Saggese, 2016). Practitioners often point the costs as barrier to the adoption of such practices, although studies show that there is a lack of knowledge about the amount of resources needed to achieve a determined level of security assurance (Yang et al., 2015; Heitzenrater and Simpson, 2016a,c; Geer, 2010).

Current models for software effort prediction do not consider the security factor and may cause inaccurate estimations, which affects the planning of required resources for software project development. A few cost models were proposed to predict effort considering security activities, but they do not take into account current software security practices and were not sufficiently validated with empirical data (Yang et al., 2015; Venson et al., 2019b). Verendel (2009) points the lack of repeated large-sample empirical studies as an obstacle to validate security quantification models.

While the amount of effort or the cost required to incorporate security practices in software development is still an open issue, studies demonstrate that this effort cannot be neglected (Chehrazi et al., 2016; Venson et al., 2019b; Heitzenrater and Simpson, 2016a). A survey with security professionals and developers on LinkedIn showed that practitioners largely use expert judgment and work breakdown methods to plan project resources, yet at the same time many projects fail to include some of the practices that will be used in the plan or even do not include any practice at all (Venson et al., 2019b). These facts raise the concern that cost estimations do not provide enough resources for secure software development, which, in addition to the lack of security culture from developers, managers and business stakeholders, contribute to the increasing number of implementation and design vulnerabilities found in software systems (Kuhn et al., 2017).

In order to support cost estimation for secure software development and cost-effectiveness evaluations, this study aims at quantifying the effects of secure software development in the development effort. The goals are to develop a scale for determining the usage degree of software security practices; and to build and validate a security cost estimation model, based on the COCOMO framework (Boehm et al., 2000),[1] using data collected from software projects in industry with varying levels of security.

With this study, we make the following contributions:

- We develop a scale to determine the degree of application of software security practices.
- We present effort multipliers for increasing levels of security practices application in software development based on security expert opinion.[2]
- We build and validate a statistical cost model based on the COCOMO framework with a data set of 1140 projects to predict effort for secure software development.
- We present effort multipliers for increasing levels of security practices application in software development based on the validated statistical cost model.

This paper is organized as follows: Section 2 presents a literature review on the topics analyzed in this study; the methodology and procedures selected are explained in Section 3; the results for the scale development and the statistical model building are reported on Section 4; Section 5 explains the findings and limitations of the research and provides recommendations for studies to follow; and, finally, Section 6, *Conclusion*, summarizes and reflects on the results obtained in this study.

[1] This framework uses cost drivers with ordinal rating scales that index into continuous interval numerical values.

[2] The term "expert" as used in this paper refers to industry experts defined as experienced professionals working in the software security industry.

2. Related research

2.1. Need for secure software development cost estimation

Secure software development can be costly because of additional processes and tools that require expertise, time, and resources from the development organization (Heitzenrater and Simpson, 2016a). Software security practices applied over the software lifecyle are pointed out as the main sources of cost for software security (Venson et al., 2019b,a).

The importance of estimating the effort and planning for security has been addressed in some studies. Peeters and Dyson (2007) raise the problem of prioritization of quality requirements, such as security, in agile development. According to them, agile's fundamental goal is to deliver software 'fit for purpose', instead of grand technical solutions, but the lack of estimation and adequate planning for quality requirements causes problems. Agile developers use to hide the costs with quality in the estimates of user stories, but this practice is harmful for protecting a software system from serious threats, which usually require more resources. As a solution, *abuser stories* (the security extension of user stories) were proposed as a way to make security requirements explicit, thus helping in planning and estimating efforts for secure software development (Peeters and Dyson, 2007). Similarly, Heitzenrater and Simpson (2016b) approach the problem of making explicit the costs with security by applying economic utility functions withing the development of *negative use cases*. *Negative use cases*, like *abuser stories*, *abuse cases*, or *misuse cases* are applied at requirements and architectural levels in order to represent scenarios where an attacker tries to use the software system for malicious purposes (McGraw, 2006; Heitzenrater and Simpson, 2016b). By integrating economic factors, negative use cases become a means of quantitative justification and provide a tradespace for dealing with resource constraints (Heitzenrater and Simpson, 2016b). However, these approaches rely on developers having proper knowledge to define the negative use cases and estimate the effort to implement the respective countermeasures.

A survey with 46 organizations at two security conferences indicated that for 23.9% of the companies, formal secure software development lifecycles are too time consuming (Geer, 2010). On the other hand, software security vendors say that the cost of implementing secure development methodologies is much less expensive than developers assume (Geer, 2010).

At the project level, stakeholders also often disagree. Heitzenrater and Simpson (2016a) observe that security professionals often consider that they do not have enough resources to deal with security in software projects. A survey with practitioners revealed a perception that it is difficult to allocate effort for including security practices in software projects, mainly due to lack of a security culture from developers, managers, and business stakeholders (Venson et al., 2019a). Further results of the survey reveal that the effort dedicated to security activities in new development projects is around 20% (median) of the total costs. However, the planning and estimation of such tasks is problematic. For more than 40% of the projects selected by the participants, not all activities or even no security activity were taken into account during planning. Also, as most of the projects applied *Expert Judgment* and *Work breakdown* as estimation techniques, this means that many projects do not receive the resources needed for properly implementing security. These examples show that there is a lack of knowledge and data about the costs of secure software development.

2.2. Approaches to estimating costs of secure software development

A systematic mapping of the literature revealed that most of the existing approaches for estimating effort to develop secure software are COCOMO II-based models (Venson et al., 2019b).

The first attempt to introduce the security factor in a software cost model was made by Reifer et al. (2003), motivated by the security risks

of incorporating Commercial Off-the-Shelf (COTS) software in critical systems. They extend COCOMO II to model the impact of security on development effort and duration, and relate it to a framework that estimates effort for COTS-based development (COCOTS). The study proposes the use of an optional cost driver for security in COCOMO II, built upon the knowledge obtained around the Common Criteria (CC) standard (Common Criteria, 2017b).

Some years later, Colbert and Boehm (2008) presented COSECMO, an extension to COCOMO II, to account for the development of secure software. COSECMO provides the total effort for the development of a new software system whose security is assured as the sum of the effort to develop security functions and the effort to assure that the system is secure. The effort is computed according to an assurance level (%Effort(AL)), defined with a new cost driver named *SECU*, whose levels are based on the Common Criteria (CC) Evaluation Assurance Levels (EAL) (Common Criteria, 2017b).

The next COCOMO-based model was proposed by Lee et al. (2014) to estimate systems in the defense domain. Their study extends the seven-step modeling methodology of COCOMO II (Boehm et al., 2000) and extracts cost factors related to the defense domain to develop a specialized software cost estimation model. Security is one amongst five factors found for the defense domain, which also include Interoperability, Hardware and software development simultaneity, Hardware emulator quality, and Hardware precedentedness.

Yang et al. (2015) reviewed previous estimation models for secure software development and proposed a model based on COCOMO II, but customized for a Chinese IT security standard. The model adds the cost driver *SECU* as a multiplier in the original COCOMO II equation. Five rating values are defined for *SECU*, according to the security degree of the software system to be estimated, which is mapped from the CC EAL. Authors conducted a mini-delphi including themselves and industry experts to establish the values for the rating scale.

Apart from COCOMO II-based models, five other studies found through a systematic mapping (Venson et al., 2019b) present approaches related to estimating costs of software security. However, except for a function points (FPA) extended model that proposes a software security characteristics formulation (Abdullah et al., 2010), all other approaches have a restrict scope of application, e.g. estimating effort to fix vulnerabilities.

Fig. 1[3] compares the models ratings for the security factor, showing that the additional effort for security can vary largely, according to the model and the level of security. Even in the same level, the differences are considerable. For example, for a high level of security, the models add 27% (COCOMO II security extensions), 20%–80% (COSECMO), 87% (Weapon systems), and 25%–50% (Secure OS software cost model) to the project effort.

Most of these approaches were not empirically validated. One study, *Secure OS software cost model* (Yang et al., 2015), verified the model through a case study. The *Weapon systems development cost model* study (Lee et al., 2014) presented an attempt to calibrate a model using data from the defense domain, however the data sample size was not sufficient to achieve statistical significance.

For Yang et al. (2015), there is a vicious spiral: since models like COCOMO II and FPA are not applicable to secure software development, estimators do not use them and, as a consequence, do not produce historical data, which prevent models from being validated or improved. Lee et al. (2014) point out that for weapon systems, the long development period hinders data collection, making it difficult to validate the model.

The validation of methods and models that estimate software security development effort with empirical data is needed so to make them useful for practitioners and researchers. With this study, we add to the existing works by applying data from 1140 software projects to develop and validate a statistical model that quantifies the effect that different security levels have on software development effort.

[3] This is an example of mapping an ordinal rating scale to an interval continuous value scale.

2.3. Determining the software security level

Any cost model intended to provide estimates for secure software development will have as input some measure[4] to indicate how much security is being considered. Some models use a count of some attribute to express the level of security (e.g. the number of security reviews performed), but they simplify assumptions over the security scope. Seeking to provide a broader scope, most of the security cost estimation models, described in Section 2.2, use the Common Criteria standard (CC) Evaluation Assurance Levels (EAL) (Common Criteria, 2017a) as a rating scale to specify the degree of security required by the application.

CC is an international standard that provides a framework to (1) specify security requirements and claims, and (2) to evaluate and certify products according to the specification (Tierney and Boswell, 2017). The Evaluation Assurance Levels (EAL1 through EAL7) define a scale for determining the level of assurance of the system during a CC security evaluation. One of the most important benefits of the framework is that it allows users to compare IT products according to the implementation of security features. Thus, the standard has been mostly used to assess and certify the security of IT Products (Kaluvuri et al., 2014).

Despite the importance of CC for Information Security, discussions around the suitability of such standard for secure software development have been raised. CC is sometimes criticized because of difficulties related to comparability, “point in time” certification, concerns for mutual recognition and the high costs of certification (Lee et al., 2016; Kaluvuri et al., 2014). Duncan and Whittington (2014) argue that compliance with standards does not necessarily leads to security, and add that a more intelligent approach is to focus on internal process and commit with a security mindset.

CC evaluations are also expensive and take time. The CC assurance is heavily focused on documentation, which must follow a specific structure, as required by the certification process, requiring a reasonable effort (Beckers et al., 2013). According to Tierney and Boswell (2017), an evaluation for EAL4 typically takes six to nine months, and higher level evaluations can take significant longer.

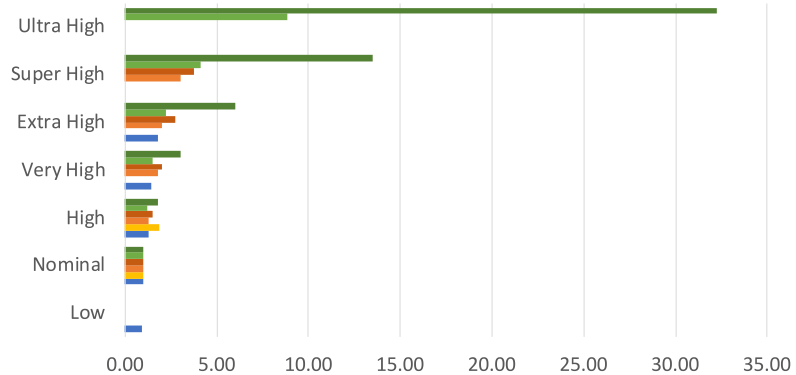
While CC represents a foundation for security evaluation of government and commercial products, who can justify a considerable budget to security, it is hardly considered by software developers who are seeking to introduce security practices in their software development process. CC is a product evaluation-focused standard that can provide some secure software development guidance. EALs are defined around the depth and rigor of design, tests and reviews of security features. However, it does not address secure software development directly as other models that were conceived around it, such as the *Microsoft Security Development Lifecyle (SDL)* (Howard and Lipner, 2006), the *Building Security in Maturity Model (BSIMM)* (Migues et al., 2019), and the *Software Assurance Maturity Model (SAMM)* (OWASP SAMM Project, 2017).

As noted, CC EALs, which are used by many cost models as a rating scale, do not offer a well fitted metric for secure software development in general. Furthermore, so far there are no objective metrics available for software security that could be used as input for cost models. Our work contributes with the development of a new rating scale, founded on the actual sources of cost for secure software development, developed according to scale development best practices.

2.4. Sources of cost in secure software development

While establishing an objective measurement for software security is still far in the horizon, the application of security practices can be

[4] Software measurement terminology is not a consensus in the fields of Engineering and Computer Science. The terms we use in this article might have different meanings in different areas.



	Low	Nominal	High	Very High	Extra High	Super High	Ultra High
COSECMO (max)		1.00	1.80	3.00	6.00	13.50	32.25
COSECMO (min)		1.00	1.20	1.50	2.25	4.13	8.81
Secure OS sw cost model (max)		1.00	1.50	2.00	2.75	3.75	
Secure OS sw cost model (min)		1.00	1.25	1.75	2.00	3.00	
Weapon systems cost model		1.00	1.87				
COCOMO II security extension	0.94	1.02	1.27	1.43	1.75		

Effort Multiplier

Fig. 1. Cost models compared.

an indicator of secure software development costs. A previous study showed that the costs of secure software development are mainly related with the application of security practices (Venson et al., 2019b). They represent 80% of the sources extracted through a systematic mapping of the literature. On the top of the list are *Perform Security Review*, which considers the inspection of the deliverables in a project, followed by *Apply Threat Modeling*, to anticipate, analyze, and document how and why attackers may attempt to misuse the software, and *Perform Security Testing*.

Many papers analyzed in that study referred to security activities established by known secure software engineering processes and standards, such as Comprehensive, Lightweight Application Security Process (CLASP), Microsoft Secure Development Lifecycle (SDL), Common Criteria for Information Technology Security Evaluation (CC), and Touchpoints.

Survey studies on the application of such practices (Morrison et al., 2017; Venson et al., 2019a) confirm they are used by practitioners. Thus, we use these practices in the development of a new rating scale to qualitatively measure secure software development with different degrees of application. This way we expect to better capture the actual development scenario in a project and, also, facilitate the security effort data collection and improve models' accuracy.

3. Methodology

This research aims at quantifying the effects of secure software development in the development effort. With this study we aim to answers two questions:

- RQ1: How to quantify the increasing levels of security practices application in software development?
  The establishment of a scale to determine the degree of application of security practices is an important step towards the data collection and calibration of the software security cost model.
- RQ2: What is the increase in development effort caused by growing levels of application of security practices?
  With data collected based on the scale, we can build a model to quantify the factors that affect software development effort, with a focus on security. The quantification of the factors allow measuring the effect that different levels of security have on software development effort.

3.1. Rating scale development

Scales are measurement instruments that combine sets of items into a composite score and are frequently developed to reveal levels of theoretical variables that cannot be directly observed (DeVellis, 2011). Methods for scale development have long been discussed in health and social sciences (DeVellis, 2011; Boateng et al., 2018), providing rigorous and tested approaches. Such methods and concepts can be applied for scale development in Software Engineering, considering the nature of the phenomenon being analyzed, which, in this case, involves development teams applying sets of security practices in different degrees.

The process to create the security rating scale was adapted from a recent primer on best practices for developing and validating scales for health, social, and behavioral research, proposed by Boateng et al. (2018). The primer presents an overall process to develop scales for measuring complex phenomena. The process has nine steps, divided into three phases, which can be selected according to the purpose of the research, the intended end-users of the scale, and the resources available.

In the first phase, items for the scale are generated and the content is validated. The second phase involves the steps to construct the scale and pre-testing it. The third and last phase is about the scale evaluation, regarding its dimensions, reliability, and validity. Phase three was not applied in this study, partly because some of the tests (dimensionality) were not necessary for the scale, and partly because of the unavailability of the required data. Although a formal evaluation, as proposed in phase three, was not possible, informal evaluations and feedback provided at the end of phase two showed that the scale fitted its purpose.

Fig. 2 illustrates the whole process, with the activities planned for each phase, adapted to this research needs and resources.

Adaptations and simplifications were made to the original process based on the characteristics of the proposed rating scale, which sometimes differ from typical scales from health and social sciences:

- The subject to be measured is a software development project.
- The property to be measured is the secure software development level, captured by the degree of application of security practices.

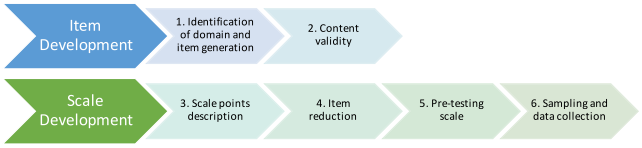


Fig. 2. Scale development phases and steps.

- The final scale consists of one aggregated item (secure software development level) composed of initial individual items (software security practices).

Further details on the adaptations are described for each phase and step description in the following subsections.

3.1.1. Item development

This phase is concerned with the content of the scale and has two steps. Step 1, *Identification of domain and item generation*, specifies the boundaries of the domain and the appropriate questions or items related. As recommended in the primer, this study defines the domain and items of the scale based on a literature review. Step 2, *Content validity*, evaluates how adequate the items are to measure the target phenomenon, regarding relevance, representativeness, and technical quality (Boateng et al., 2018).

3.1.2. Scale development

Phase 2, the scale construction, was developed in four steps (Steps 3 to 6).

Step 3, *Scale points description* is not in the primer, but this activity was needed for this research because the proposed scale to determine the level of security in software development requires more than a numeric item as response (as a Likert scale). Each point in the scale needs a proper specification that describes the intensity and rigor of the execution of a set of software security practices.

Given the difficulties in collecting data from software practitioners, step 4, *Item reduction*, originally the fifth step, after pre-testing questions and survey administration, was brought forward to prune the instrument, before presenting it to the scale users. The purpose of this step is to remove the less significant items of the scale, ensuring that only the items that contribute to the measurement are retained. The criteria established to keep only the most relevant practices in the scale was to evaluate the degree of contribution of the practice's scale points description to distinguish levels of security practices application. For the practices to be removed, the nature of the activities involved should not change significantly from a basic level of security to a high level of security, except for the descriptor for attribute level.

Step 5, *Pre-testing scale*, involves ensuring that the items are meaningful to the target population, minimizing misunderstandings and measurement error (Boateng et al., 2018). Two components are examined: the extent to which the questions reflect the domain being studied and the extent to which answers produce valid measurements. To evaluate if the rating scale reflects the domain being studied, the focus group technique was chosen. This is one of the approaches suggested by Boateng et al. (2018) for this step. Also, to evaluate if the rating scale is able to produce valid measurements, a Wideband Delphi (Boehm, 1981) session was selected as strategy. Both approaches, the focus group discussion and the Wideband Delphi session, were planned as two workshops to happen in the annual research review event organized by the University of Southern California (USC). This event is a good opportunity to conduct these studies because it gathers researchers and industry affiliates who are potential users of the proposed model. Also, the attendees of this event are used to Wideband Delphi sessions.

Step 6, *Sampling and data collection*, involves using the scale to collect data. The secure software development level is one among other variables to be collected for a set of software development projects. The sample is composed by companies that are interested in the construction of the model.

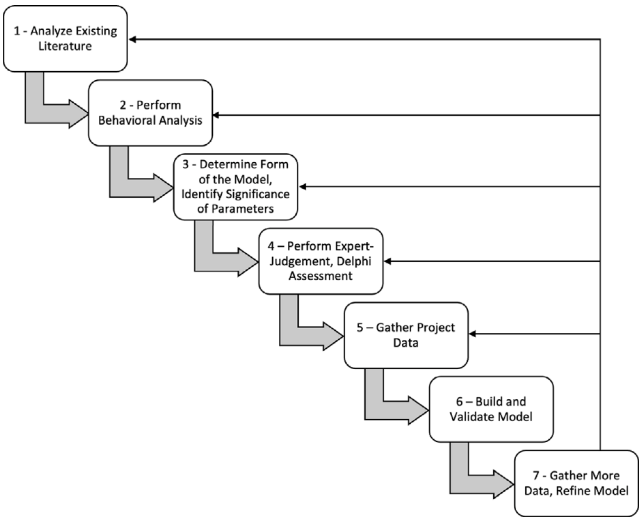


Fig. 3. Modeling methodology.
Source: Adapted from Boehm et al. (2000).

3.2. Statistical model building

Research question 2 was addressed by establishing a parametric model to explain the increase in development effort caused by specific degrees of security practices application in software development.

Data was collected from industry experts and software projects and analyzed to quantify the factors that affect software development effort, including Security.

The model followed previous works on the COCOMO II statistical model-building process (Chulani, 2001; Boehm et al., 2000). The modeling methodology comprises seven steps. The first four steps overlap somewhat with the rating scale development process. Fig. 3 depicts the process, adapted from Boehm et al. (2000).

The next subsections describe how each step was planned, in order to answer the research question 2.

3.2.1. Analyze existing literature

The existing literature around costing secure software was broadly analyzed in a previous study (Venson et al., 2019b). Existing cost models that address security were surveyed, and sources of cost were extracted from the literature. The results of this review revealed the issues with the existing models and provided insights on how to improve them by the means of a new rating scale, further analysis, and empirical validation.

3.2.2. Perform behavioral analysis

A behavioral analysis was explored in the survey study with practitioners (Venson et al., 2019a). The data gathered from the participants, most of them security experts working on software projects, showed that the introduction of security-related tasks in the software development process affects the overall effort. Furthermore, it was found that projects apply security practices with different frequencies and time, suggesting that it may vary according to the product characteristics, software lifecycle and team structure.

3.2.3. Determine form of model, identify relative significance of parameters

The original COCOMO II model has the following mathematical form (Boehm et al., 2000):

$$PM = A \times Size^{E} \times \prod_{i=1}^{n} EM_i \quad (1)$$

where:

PM = Effort in person-month

A = Effort coefficient that can be calibrated

Size = Represents the software size

EM = Effort Multipliers

E = Scaling Exponent for Effort

In Eq. (1), E is the exponent related to scale factors:

$$E = B + 0.01 \times \sum_{j=1}^{5} SF_j \quad (2)$$

where:

B = Scaling base-exponent for Effort that can be calibrated

SF = Scale Factors

COCOMO II uses the product of 17 Cost Driver Effort Multipliers to increase or decrease estimated effort. An example of a Cost Driver is Language and Tool Experience (LTEX). LTEX has five rating levels: Very Low, Low, Nominal, High, and Very High. The description for each rating level is provided in the model documentation. Each rating level has an associated Effort Multiplier (EM): Very Low = 1.20, Low = 1.09, Nominal always = 1.0 for all Cost drivers, High = 0.91, and Very High = 0.81. When a cost driver rating level is selected, the associated EM is used in the mathematical model.

A new parameter, SECU, representing the effort multiplier for the increasing levels of security practices application in software development, was introduced in the model, modifying equation (1) to:

$$PM = A \times Size^{E} \times SECU \times \prod_{i=1}^{n} EM_i \quad (3)$$

Note that the goal of the model building is to establish SECU multipliers considering their relationship with other cost drivers for the development effort.

This step also involves the determination of the rating scale and the definition of the meaning for each point in the scale, according to the project tasks. This was deemed in steps 3 and 4 of the scale development, as described in Section 3.1.2.

3.2.4. Perform expert judgment, Delphi assessment

Two panels with experts were planned. One session of Delphi to test the security scale and collect initial estimates from cost model experts, involving researchers and industry affiliates of the CSSE annual research review, as reported in step 5 (*Pre-testing scale*) of the scale development.

A second Delphi was planned online with professionals that work in the Software Security industry, recruited from the LinkedIn Software Security Group.[5] Given the geographically diverse localization of the invitees, this data collection was conducted as an online Delphi study (e-Delphi) (Donohoe et al., 2012; Khodyakov et al., 2020; Morton et al., 2017).

The process was planned and executed as illustrated in Fig. 4. Initially, the facilitator sent an invitation request to the security experts, providing an overview of the study and a presentation of the rating scale. Next, the participants provided their initial estimates and comments for the rating scale by filling an online form. After the established deadline for round 1, the researchers consolidated and discussed the results, prepared and sent back a customized report for each participant. The report contained a summary of the compiled results, with histograms showing the answers' distribution and the participant's own estimates, besides elucidation of common questions and assumptions made by other experts. In round 2, participants were invited again to discuss the results and re-estimate the effort range, based on the feedback from round 1. Once again, the results were compiled and sent them back to all participants.

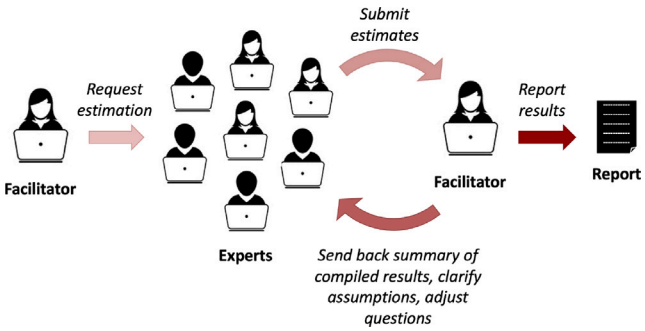


Fig. 4. The online Delphi process.

In each round the participants were asked to provide a *productivity range* for the three groups of the scale — Requirements & Design, Coding & Tools, and Verification & Validation. The productivity range is defined as the ratio between the highest level (Ultra High) and the lowest level of the scale (Nominal). For example, a productivity range of 3 means that to apply the practices in the Ultra High level, it would take 3 times more effort compared to the Nominal level (no explicit security-related activity), an increase of 300% effort.

Participants were also asked to consider a set of assumptions during estimation:

- Assume that there is a security team supporting the developers and performing specialized tasks (e.g. penetrating testing, managing external penetration tests, etc.).
- Assume that a hybrid security practices approach is taken (manual and automated).
- Assume that other aspects that affect the productivity range (team experience, product complexity, reuse, etc.) will be captured by factors other than the security scale.
- Assume that the productivity range does not include initial setup effort for security tools and infrastructure.
- Assume that the security measures are applied at the same level across the groups.
- Assume that the productivity range includes only the security practices applied (code review, penetration testing, threat modeling, etc.), excluding the development of the security features.

Considering that the costs with security tend to present an exponential growth as the security level is increased (Böhme, 2010), the scale points values were calculated by applying exponential grow equations.

3.2.5. Gather project data

The instrument for collecting project data points was in the form of a spreadsheet that was distributed to industry partners to provide the security ratings, software size, effort, and schedule data. The instrument also collects the attributes that provide contextual information about the project and ratings for other factors that drive effort besides security.

The instrument was reviewed by estimation experts before being sent out to participants.

[5] https://www.linkedin.com/groups/3027331/

3.2.6. Build and validate model

This is the step where the model is statistically built based on the data collected from project data.

It starts with the examination of the data collected in the previous steps to verify if the observations were properly recorded, identify possible subgroup behavior, or other issues that could indicate a need to review the data collected or change the form of the model.

One perspective analyzed in the model building was the software domain. As indicated by Menzies et al. (2017), different software application domains appear to establish different development modes, which may drive costs and schedule of projects differently. A study from Rosa et al. (2017) showed that the introduction of the domain variable improves effort prediction in early phase cost models.

Having formulated the problem, the data was then fitted into the model, generating a calibration of the model parameters. The calibration involves (1) linearizing Eq. (3) by taking logarithms on both sides of the equation; and (2) applying multiple linear regression (MLR) to find the coefficients of the equation.

After estimating the parameters of the model, the coefficients calculated need to be examined to verify their magnitude (t-value) and to check if they indicate the expected behavior (correct signs). The remediation may involve changes in the model parameters or the data set (removing outliers, for example).

Deriving the coefficient for the $SECU$ predictor, was the main focus of this research. The value of this coefficient indicates the degree of influence that the security parameter has on the projects' effort. A coefficient close to zero would indicate that $SECU$ does not influence effort. Besides that, the t-value (ratio between the coefficient estimated value and corresponding standard error) indicates the statistical significance of the $SECU$ parameter. The higher the t-value, the stronger the statistical significance of the predictor variable.

Once the model was considered adequate for the data and parameters estimated, the fit was evaluated to verify if it can generate good predictions involving different levels of required security for the projects. The 'goodness of fit' of the model to the data in the MLR was evaluated by the adjusted R-squared value and the standard error. To test the model accuracy, the Validation Set approach and k-fold Cross Validation were employed.

As there is no standard method to select k in k-fold Cross Validation, we chose k = 5 as it is a commonly used value and would make the validation set with a similar size as the 75%–25% values of the Validation Set approach. The percentage of predictions within 25 percent of the actuals PRED(25) was also calculated.

3.2.7. Gather more data, refine model

The modeling methodology, as illustrated in Fig. 3, is iterative. As new data becomes available, the model can be continually refined by reviewing any of the previous steps of the methodology.

4. Results

In this section we first present the resulting scale for increasing levels of security practices application in software development, including the estimates for the productivity range obtained through the online Delphi, and the preliminary results of effort multipliers for the security factor; next, the second subsection presents the results for the statistical model building.

4.1. Secure software development scale

The domain of the scale was identified with a systematic mapping of the literature, which revealed that previous cost models for security effort used the Evaluation Assurance Levels (EAL) from the Common Criteria (CC) standard as a rating scale (Venson et al., 2019b). However, as observed in Section 2.3, the EAL scale is not completely adherent to the factors that drive security efforts in software development projects. No other scale to determine secure software development level was found.

The scale items' list initially comprised 13 practices, extracted from the literature (Morrison et al., 2017; Venson et al., 2019b). Then, a survey with software security experts recruited from LinkedIn tested the *Content validity* for the selected items (Venson et al., 2019a). The participants of the survey are potential end-users of the rating scale. In the questionnaire, the respondents answered for each item (security practice) what was the frequency of use and the effort required to apply it each time. Results of the survey showed that all 13 practices are frequently used by professionals.

Those practices were described in increasing levels, then pruned and summarized as explained in Section 3. The description of the scale points for each item was performed based on descriptors for attribute levels of the COCOMO II model, combined with existing models that reflect the maturity of security practices application. Cost drivers' levels in COCOMO II range from the *Extra Low* level to the *Extra High* level with specific modifiers and descriptors that were used to classify the presence and degree of an attribute in the software product to be developed. For example, the *Extra Low* level modifier is *Extremely* and the descriptors are *(Easy, Little, Simple, Informal, Casual)*. BSIMM (Migues et al., 2019) and SAMM (OWASP SAMM Project, 2017) models were used to help describe the increasing levels of security practices' application. It is important to notice the distinct meanings of *maturity of practices* and *degree of use of the practices*. For example, one organization may be mature in some security practices, but it may apply these practices with different frequency and rigor in its projects, according to the security risks involved. The description of practices in increasing maturity levels, thus, cannot be directly mapped to the increasing degrees of usage but can help understand how sophistication is introduced in the process as the security needs to step up. As a result, each of the items in the rating scale is composed of a combination of the COCOMO II attribute level descriptors and the description of the practices in increasing levels of maturity.

Table 1 presents the Secure Software Development Rating Scale obtained at the end of the whole process. The scale captures the level of usage of software security practices in the project, which are divided into three groups: (1) Security Requirements and Design, (2) Secure Coding and Security Tools, and (3) Security Verification and Validation. Each level, from *Nominal* to *Ultra High*, represents an increasing degree of rigor and intensity for each group of practices.

This is the latest version of the table, that has been adjusted during focus groups and with the feedback provided by participants of the Online Delphi study.

4.1.1. Expert estimates

Round 1 of the Delphi was online for 10 days and we received responses from 17 industry security experts. In round 2, which lasted 12 days, 14 out of the 17 participants responded with new estimates. The reported average of experience in secure software development was 10.88 years and 11.05 years average in effort estimation.

Table 2 presents the summary results of productivity range[6] estimates for each of the three security practices' group - *Requirements & Design*, *Coding & Tools*, and *Verification & Validation*. Comparing Round 1 and Round 2, we can observe that the average and median productivity range for all groups were considerably lower in the second estimation. The Coefficients of Variation (CV) decreased in Round 2 for Secure Coding and Tools and Security V&V, though still high.

Fig. 5 presents the histogram of the estimates for the Requirements and Design group for Round 2. The distribution is skewed right, with

[6] Productivity ranges show the amount effort is adjusted by the model. A value of 1.0 means no increase in effort. A value of 2.28 means an increase in effort of 128%.

Table 1
Secure software development levels.

Level	Security requirements and design	Secure coding and security tools	Security verification & validation
Nominal	None	No secure coding and no use of static analysis tool.	None
High	Basic analysis to identify security requirements. Basic threat modeling.	Basic vulnerabilities applicable to the software will be prevented with secure coding standards and/or detected through basic use of static analysis tools.	Basic adversarial testing and security code review. Basic penetration testing. Security VV activities conducted within the project.
Very High	Additional analysis to identify security requirements such as audit/log, cryptography, etc. Moderate threat modeling.	Known and critical vulnerabilities applicable to the software will be prevented with secure coding standards and/or detected through routine use of static analysis tools.	Moderate adversarial testing and security code review. Routine penetration testing. Security V&V activities conducted by an independent group.
Extra High	Thorough analysis to identify security requirements, advanced secure-by-design needs. Threat modeling with specific attack strategies.	Extensive list of vulnerabilities and weaknesses applicable to the software will be prevented with secure coding standards and/or detected through extensive use of static analysis and black-box tools.	Extensive adversarial testing and security design/code review. Frequent and specialized penetration testing. Security V&V activities conducted by an independent group at the organizational level.
Ultra High	Extensive analysis to identify security requirements, including off-nominal cases, container-based approaches for advanced security features development. Rigorous threat modeling.	Very extensive list of vulnerabilities and weaknesses applicable to the software will be prevented with secure coding standards and/or detected through rigorous use of static analysis and black-box security testing tools with tailored rules. Employ formal methods in coding.	Rigorous adversarial testing and security design/code review. Exhaustive deep-dive analysis penetration testing. Use of formal verification and custom developed V&V tools. Security V&V activities conducted by an outside certified company.

Table 2
Productivity range estimation statistics for all groups in round 1 and round 2.

	Round 1			Round 2		
	Req & des	Coding & tools	Ver & val	Req & des	Coding & tools	Ver & val
Average	2.28	2.71	5.45	1.96	2.05	2.56
Median	2.00	2.30	3.00	1.50	1.40	1.75
SD	1.04	2.00	11.53	1.09	1.19	2.34
CV	46%	74%	212%	56%	58%	91%

Fig. 5. Productivity range distribution for the Requirements and Design group.

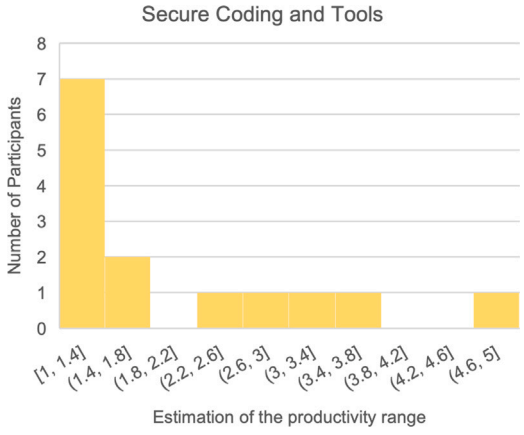


Fig. 6. Productivity range distribution for the Coding and Tools group.

estimates concentrated between 1 and 2.05. The range is between 1 and 4.5, with an average of 1.957 and a standard deviation of 1.093.

For the Coding and Tools group, Fig. 6, the distribution of the range estimates on the second round is also skewed right, with 7 estimates between 1 and 1.4. The average is 2.046, with a standard deviation of 1.193. The range is between 1 and 5.

The distribution of the range estimates for the Verification and Validation, Fig. 7, is strongly skewed right, with an average of 2.561. The standard deviation is 2.335 and the range is between 1 and 10.

By multiplying the averages of the three groups, we calculate the productivity range for the security scale as 10.256. However, considering the skewness of the distributions, it seems more appropriate to use the median to obtain the productivity range, which in this case is 3.675 (1.5 * 1.4 * 1.75). This means that a project at level *Ultra High* would cost 3.674 times more to develop than a project at level *Nominal*.

Distributing the productivity range over the levels using an exponential growth curve, we can calculate the effort multipliers for each level. The resulting multipliers are presented in Fig. 8.[7] According to these estimates for the scale, a project applying security practices of the High level would require 38% of additional effort, while a project

[7] The values in this chart are a provisional mapping of continuous numerical values to rating levels. The values will be adjusted and validated with real-world data.

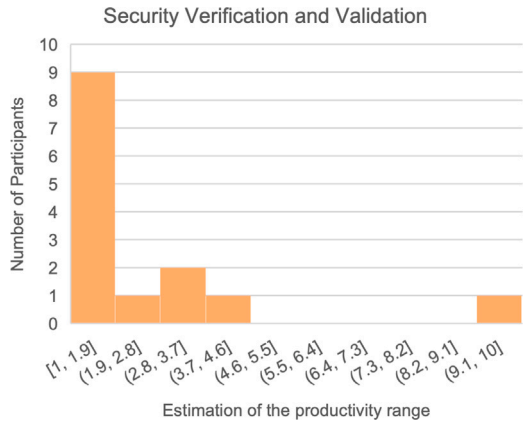


Fig. 7. Productivity range distribution for the V&V group.

Fig. 8. Effort multipliers per security level calculated from the productivity range.

applying security practices of the Ultra High level would require 367% of additional effort.

4.2. Statistical cost model for secure software development

We start this subsection by describing the data set and the collinearity test of the candidate predictors for the Multiple Linear Regression (MLR). Next, the statistical model is defined and the Required Security (SECU) predictor is presented. The following sections present the regression results, the model validation, the resulting model, and the values obtained for the SECU multiplier.

4.2.1. Data set description

The data set used to calibrate the cost model contains 1140 maintenance projects, performed during 2019 and 2020, by two companies in Brazil: a large banking institution, and a telecommunications company.

The banking institution uses waterfall as their main software development lifecycle. They apply statistical process control for software development processes and have about 20 years of experience in measuring function points. The telecom company employs agile methods and is less experienced in measuring projects with function points.

All projects in this data set use function points (FP) as a software size metric. Because they are maintenance projects, the software size is a composition of added, changed, and deleted data functions and transaction functions. Changed and deleted functions are deflated by a factor,[8] representing the impact on effort for changing the software according to the type of function.

Table 3
Data set summary statistics.

Variable	Mean	SD	Median	Min	Max
Size (FP)	80	155	38	1.18	2807
Effort (h)	1048	1774	525	1.00	22,024
Effort (PM)	7	12	4	0.01	145
Prod. (h/PF)	17	32	13	0.25	680

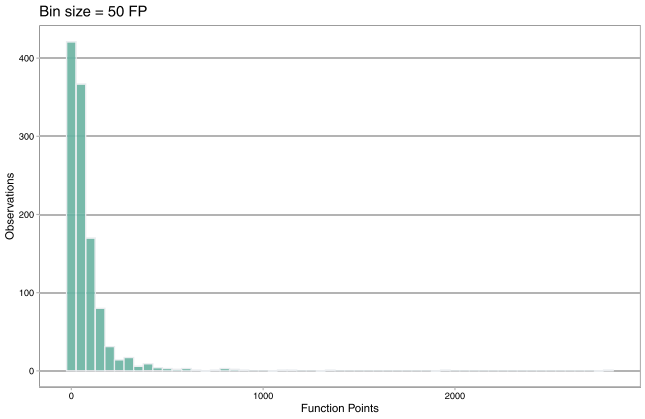


Fig. 9. Function points distribution.

Eqs. (4) to (6) define the composition of the size metric used for all observations in the data set:

$$Size = FP_{DataFunctions} + FP_{TransactionFunctions} \quad (4)$$

$$FP_{DataFunctions} = DF_{Added} + DF_{Modified} * 0.4 + DF_{Deleted} * 0.3 \quad (5)$$

$$FP_{TransactionFunctions} = TF_{Added} + TF_{Modified} * 0.6 + TF_{Deleted} * 0.3 \quad (6)$$

where DF represents the function points for the Data Functions and TF represents the function points for the Transaction Functions.

Table 3 presents the summary statistics for the main variables of the data set.

The projects' size ranges from 1 to 2807 FP, with an average of 80, a median of 38, and a standard deviation of 155 FP. Fig. 9 illustrates how the distribution of FP is strongly right-skewed.

Projects' effort ranges from 0.01 Person-Month (PM) or 1 h to 145 PM or 22,024 h, with an average of 7 PM, a median of 3.45 PM, and a standard deviation of 12 PM. The effort in hours is divided by 152 to convert it to PM. Fig. 10 shows the distribution of the effort for the data set, also strongly right-skewed.

The scatter plot of Fig. 11 shows the relationship between size and effort. Given the skewness of the variables, both axis values are presented in the log10 scale. As it can be observed, the telecom projects are smaller in size and have a higher variation to effort when compared to the banking projects. The distinct contexts of software process maturity and software development lifecycles may contribute to the differences in the plot.

The cost drivers measured in the data set are presented in Table 4. They are a subset of the cost drivers established by the COCOMO II model, with the addition of the Required Software Security (SECU) cost driver, which is defined according to the scale developed.

[8] These factors are sometimes used by industry to account for reduced effort required to modify or delete function points when compared to adding a function point. Unfortunately, the data did not separate out added, modified, and deleted function points. An independent verification of the factors was not possible.

Table 4
Cost drivers used in the data set.

Code	Name	Description
SECU	Required Software Security	This driver captures the degree of application of software security practices.
FAIL	Impact of Software Failure	This is the measure of the extent to which the software must perform its intended function over a period of time.
CPLX	Product Complexity	The complexity of the product is characterized by the operations (control, computational, device-dependent, data management, and user interface management) performed.
PLAT	Platform Constraints	This driver captures the limitations placed on the platform's capacity such as execution time, primary/secondary storage, communications bandwidth, battery power, and others.
PVOL	Platform Volatility	The targeted platform may still be evolving while the software application is being developed. This driver captures the impact of the instability of the targeted platform resulting in additional/increased work.
APEX	Application Domain Experience	This driver captures the average level of application domain experience of the project team developing the software system or subsystem.
LTEX	Language and Tool Experience	This is a measure of the level of programming language and software tool experience of the project team developing the software system or subsystem.
PLEX	Platform Experience	This driver recognizes the importance of understanding the target platform.
TOOL	Use of Software Tools	This driver rates the use of software development tools using three characteristics.
SITE	Multisite Development	The multisite development effects on effort are significant. Determining its cost driver rating involves the assessment of collocation and communication.

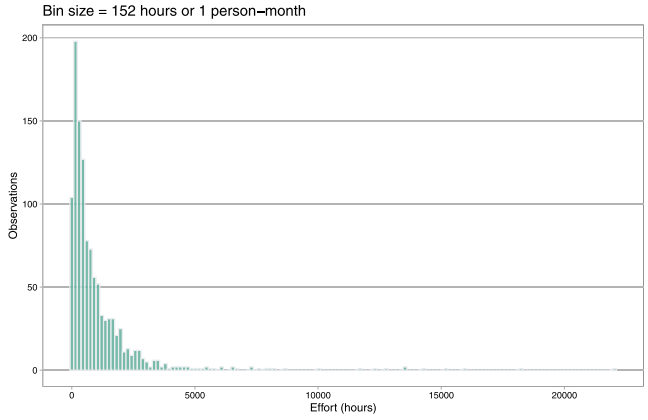


Fig. 10. Effort distribution.

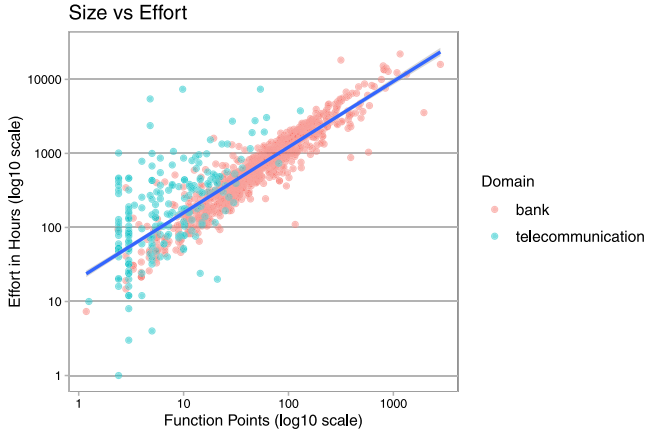


Fig. 11. Relationship between size and effort.

Table 5 shows the summary statistics for the cost drivers rated in the data set. The SECU cost driver covers the Nominal and High levels, respectively, 3 and 4.

The data set also presents the security effort for 120 projects, whose practices were classified at the level HIGH according to Table 1. This variable accounts for the effort spent in such security practices. The security effort ranges from 6.9 h to 1386.1 h, with an average of 183.68 h, a median of 98.15 h, and a standard deviation of 228.32 h. Fig. 12 shows the distribution of the security effort.

Table 5
Summary statistics for size and effort multipliers predictors.

Driver	Mean	SD	Min	Median	Max
FP	83.52	158.20	1.18	42.35	2806.99
SECU	1.03	0.09	1.00	1.00	1.28
FAIL	1.05	0.08	0.82	1.10	1.26
CPLX	0.96	0.07	0.73	1.00	1.34
PLAT	1.05	0.10	1.00	1.00	1.54
PVOL	1.02	0.08	0.87	1.00	1.30
APEX	1.04	0.08	0.88	1.00	1.22
LTEX	0.94	0.06	0.84	0.91	1.09
PLEX	0.94	0.06	0.85	0.91	1.19
TOOL	1.02	0.06	0.78	1.00	1.17
SITE	0.93	0.03	0.80	0.93	1.00

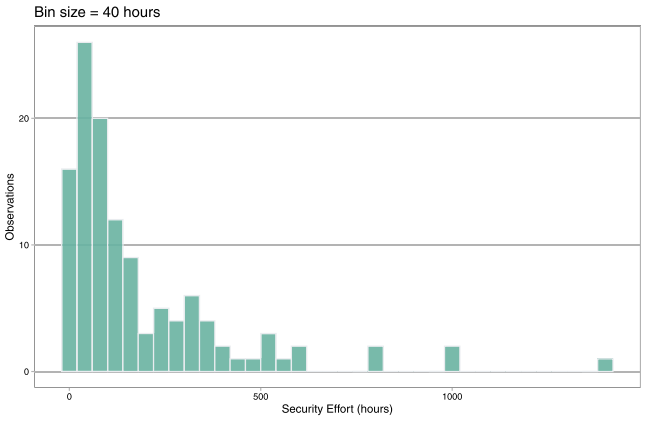


Fig. 12. Security effort distribution.

4.2.2. Collinearity test results

All predictors – the size and the cost drivers variables – were tested for multicollinearity as it can negatively affect the results of the regression.

The pairs Language and Tools Experience (LTEX) and Platform Experience (PLEX), Language and Tools Experience (LTEX) and Application Domain Experience (APEX), and Application Domain Experience (APEX) and Platform Experience (PLEX) presented strong correlations, equal or greater than 0.9. As recommended in the literature, the correlated cost drivers were aggregated into one predictor, which was named Experience (EXPE), by applying the geometric mean.

The correlogram with the EXPE predictor replacing LTEX, PLEX and APEX is showed in Fig. 13. This set of variables has only weak

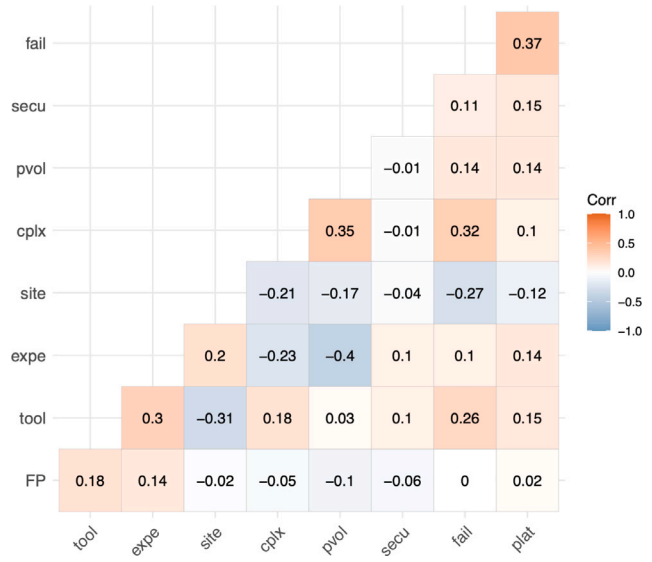


Fig. 13. Pairwise correlation for the predictors with the new aggregate cost driver EXPE.

correlations among the predictors and, therefore, was used for the model building.

For the Required Software Security (SECU) cost driver, no significant association with other variables was found. The maximum correlation value was 0.154, with the Platform Constraints (PLAT) predictor.

4.2.3. Model definition

Based on the data set analysis, the multiple regression model equation was defined as:

$$\widehat{Effort} = \beta_0 \times Size^{\beta_1} \times SECU^{\beta_2} \times FAIL^{\beta_3} \times CPLX^{\beta_4} \times PLAT^{\beta_5} \times PVOL^{\beta_6} \times EXPE^{\beta_7} \times TOOL^{\beta_8} \times SITE^{\beta_9} \times \omega \quad (7)$$

β_0 = Coefficient to be estimated for constant A (Intercept)

Size = Function Points Count of the project (fsm_count)

β_1 to β_9 = Coefficients to be estimated for the size and cost drivers

ω = The log-normally distributed error term

The coefficient for the scale exponent for effort (E) was not included in the model equation because the ratings for these cost drivers in the data set were all nominal (factor = 1) and would not affect the results.

4.2.4. Multiple regression results

The results of the multiple linear regression for the model based on Eq. (7) are presented in Table 6.[9] The model fitness information is showed in Table 7. The *p*-value of the F-statistics is $< 2.2e-16$, which is highly significant and shows that there are predictor variables significantly related to the effort.

The R^2 is 0.838 and the Adjusted R^2 is 0.837, indicating that 84% of the variance in the effort can be explained by the variables in the model. The analysis of the t-statistics for the coefficients in Table 6 reveals that, except for SITE, all other predictor variables have a significant association with the dependent variable effort. The *t*-value measures how many standard errors the coefficient is away from zero, thus the higher the *t*-value, the greater the confidence in the coefficient as a predictor.

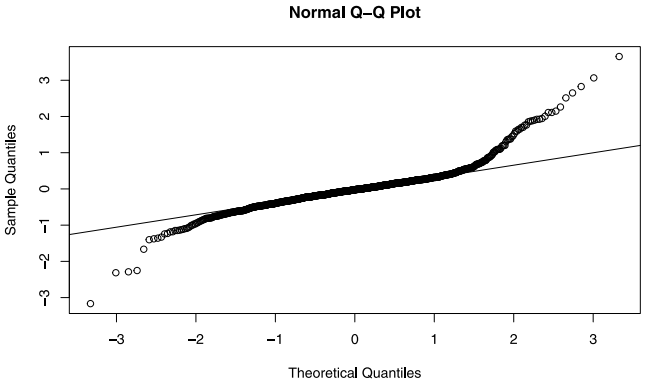


Fig. 14. Standardized Residual plot for the model.

The intercept coefficient is 2.602 and back-transforming it from natural log to the original scale ($e^{2.602}$) results in 13.49. This means that, without considering the other parameters, it takes 13.49 h to develop one Function Point.

The estimated coefficient for the Required Software Security (SECU) parameter is 1.111, with a significant *t*-value of 5.201. Also, the *p*-value for the SECU predictor is very low, supporting the evidence that the SECU cost driver has an effect on the resulting effort.

The Multisite Development (SITE) predictor presented a negative coefficient with high *p*-value. The negative value indicates that this parameter, that represents the impact of collocation and communication on effort, had a opposite behavior that was expected by the model. However, the *p*-value is large (0.207), indicating also that SITE is not significant to explain the resulting effort in this sample. One possible explanation for this result is that as most of the projects in the sample are small (the median for effort is 3.45 person-month), they require small teams for which different communication and collocation levels may not impact the productivity.

The residual quantile–quantile (Q-Q) plot presented in Fig. 14 shows the approximately normal distribution of the errors. Most residual points fall along the normality line in the plot. The points falling off the line at the extreme ends indicate that the distribution has heavy tails.

4.2.5. Outliers detection and removal

Outliers in a data set may have a considerable impact on the fitness of a model as they introduce bias and affect predictions. Two variables were analyzed to check for outliers — productivity and security effort ratio.

The projects' productivity[10] is given by the quotient between the effort and the size variables and, in this data set, is expressed in hours per Function Point (h/FP). In the data set, the productivity ranges from 0.25 h/PF to 680 h/PF, with an average of 16.9 h/FP and a median of 12.8 h/FP.

Another variable in the data set that presented extreme values is the security effort ratio. This variable is calculated as the effort spent with security in the project divided by the development effort (without the security effort). The security effort ratio ranges from 0.008 to 22.21, with an average of 0.87 and a median of 0.23.

Considering that the extreme values of productivity and security effort ratio could affect the accuracy of the model, an analysis of the outliers was performed. A well-known rule to detect outliers was defined by Tukey (1977), which states that the outliers are values

[9] The model in equation 7 is non-linear. Using a log-transformation on the model, it becomes a linear model allowing the use of multiple linear regression, i.e., log(Effort) = B0 + B1log(Size) + B2log(SECU) + ...

[10] The term productivity is a derived measure that can also be referred as unitary effort or unit effort depending on its usage. In this paper productivity is defined as Effort Hours per Function Point (h/FP).

Table 6
Coefficients for the model (log values).

Term	Estimate	Std.Error	*t*-value	*p*-value		Conf-low	Conf-high
(Intercept)	2.602	0.069	37.887	< 2e−16	(***)	2.467	2.736
log(fsm_count)	0.976	0.014	71.823	< 2e−16	(***)	0.949	1.003
log(fail)	1.201	0.241	4.985	7.18e−07	(***)	0.728	1.674
log(cplx)	0.869	0.233	3.736	0.000197	(***)	0.413	1.326
log(secu)	1.111	0.214	5.201	2.35e−07	(***)	0.692	1.531
log(plat)	0.587	0.215	2.727	0.006490	(**)	0.165	1.009
log(pvol)	0.657	0.230	2.855	0.004383	(**)	0.205	1.108
log(expe)	3.189	0.315	10.108	< 2e−16	(***)	2.570	3.808
log(tool)	3.243	0.322	10.061	< 2e−16	(***)	2.611	3.875
log(site)	−0.654	0.518	−1.264	0.206588		−1.670	0.361

Signif. codes: 0 '(***)' 0.001 '(**)' 0.01 '(*)' 0.05 '(.)' 0.1 ' ' 1.

Table 7
Model fitness.

R^2	Adj. R^2	RSE	F-statistic	*P*-value	DF	DF Residual	N Obs
0.838	0.837	0.536	650.1	< 2.2e−16	9	1130	1140

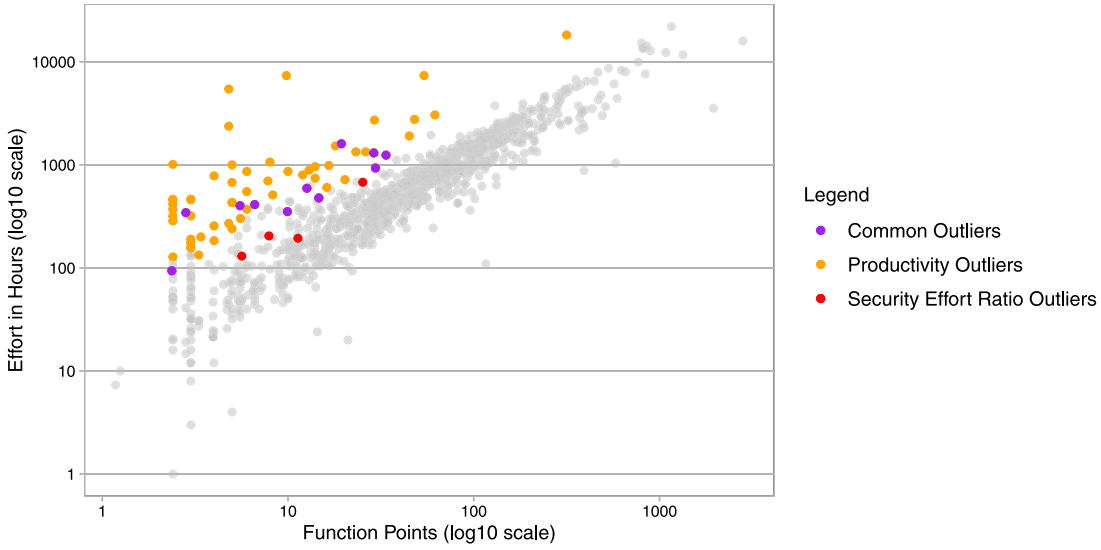


Fig. 15. Scatter plot of size vs effort with outliers highlighted.

more than 1.5 times the interquartile range (IQR) from the quartiles — either below $Q1 - 1.5 \times IQR$, or above $Q3 + 1.5 \times IQR$. Hoaglin and Iglewicz (1987) analyzed the performance of the parameter $k = 1.5$ established by Tukey and determined the probability that a sample with n observations contains no observations beyond the cutoffs using the probabilities of 90% and 95%. The study showed that for an n above 10, the median value for k is 2.1 for 90% and 2.3 for 95% probabilities that no observations are beyond the cutoffs.

The parameter $k = 2.3$ was then applied to label the outliers for the variables productivity and security effort ratio in the data set. The upper range for the productivity variable was calculated as 30.71 h/PF and 64 observations were labeled as outliers. For the security effort ratio variable, the upper range was defined as 1.74, and 15 observations were identified as outliers. In total, 68 observations out of the 1140 observations were classified as outliers, as 11 observations were commonly labeled by both the productivity and the effort security ratio rules. Fig. 15 shows the position of the outliers in the Size versus Effort scatter plot.

A new version of the data set with the remaining 1,072 observations, named *Filtered Data set* was created and used to calibrate a new version of the model.

4.2.6. Multiple regression results for the filtered data set

The results of the coefficients for the model using the Filtered Data set are presented in Table 8. The R^2 and the adjusted R^2, both rounded to 0.93, were improved with the Filtered Data set-based model, keeping a strong *p*-value. This means that 93% of the variance in effort can be explained by the variables in the model, compared to the 84% obtained with the complete version of the data set.

These results are also reflected in more significant p-values for the predictors Platform Constraints (PLAT) and Platform Volatility (PVOL), whose p-values had significance < 0.01 in the original version of the data set. The t-values of the coefficients confirm that the predictors have a significant association with the effort. The only exception was Multisite Development (SITE) parameter, whose *p*-value remained not significant. The SITE parameter was then removed from the equation.

When comparing the coefficients of the model based on the Filtered Data set (Table 8) with the coefficients of the model based on the Complete Data set (Table 6), it can be observed that the removal of the outliers caused important changes in the coefficient values. The new intercept coefficient (2.328) when back-transformed to the original scale ($e^{2.328}$) results in 10.26 h/FP, a lower value when compared to the 13.49 h/PF in the previous model. This was expected as the outliers' analysis revealed extreme values for productivity, such as the maximum value of 680 h/PF, which was removed as the upper range for the productivity variable as defined at 30.71 h/PF.

A similar phenomenon occurred to the Required Software Security (SECU) predictor. The new coefficient for SECU, 0.707, is lower than the previous value of 1.111. The analysis of the security effort ratio

Table 8
Model coefficients for the Filtered Data set (log values).

Term	Estimate	Std.Error	Statistic	*P*-value		Conf-low	Conf-high
(Intercept)	2.328	0.038	61.900	< 2e−16	(***)	2.254	2.401
log(fsm_count)	1.039	0.010	109.319	< 2e−16	(***)	1.021	1.058
log(fail)	1.386	0.165	8.415	< 2e−16	(***)	1.063	1.709
log(cplx)	0.920	0.160	5.759	1.11e−08	(***)	0.606	1.233
log(secu)	0.707	0.154	4.575	5.33e−06	(***)	0.404	1.010
log(plat)	0.563	0.152	3.695	0.000231	(***)	0.264	0.862
log(pvol)	0.607	0.162	3.742	0.000193	(***)	0.289	0.925
log(expe)	2.413	0.215	11.209	< 2e−16	(***)	1.991	2.836
log(tool)	2.166	0.220	9.864	< 2e−16	(***)	1.735	2.597

Signif. codes: 0 '(***)' 0.001 '(**)' 0.01 '(*)' 0.05 '(.)' 0.1 ' ' 1.

Table 9
Model fitness for the two versions of the data set.

Data set	R^2	Adj. R^2	RSE	F-statistic	*P*-value	DF	DF Residual	N Obs
Complete	0.838	0.837	0.536	650.1	< 2.2e−16	9	1130	1140
Filtered	0.927	0.926	0.364	1678.9	< 2.2e−16	9	1063	1072

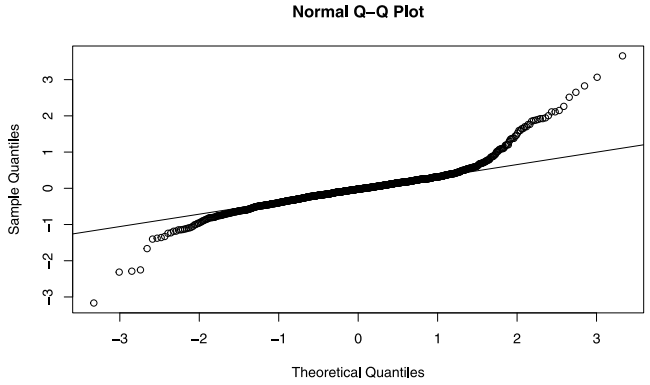


Fig. 16. Standardized Residual plot for the Filtered Data set-based model.

Table 10
Model validation results.

Data set	Approach	R2	MMRE	PRED25	PRED40
Complete	Validation Set	0.900	0.358	50.4%	68.7%
	5-fold Cross Validation	0.693	0.393	53.7%	71.1%
Filtered	Validation Set	0.910	0.296	63.1%	79.9%
	5-fold Cross Validation	0.739	0.291	64.2%	82.2%

Table 11
New cost drivers' effort multiplier values.

Cost driver	VL	L	N	H	VH	XH	UH	PR
SECU			1.00	1.19	1.42	1.69	2.02	2.02
FAIL	0.76	0.89	1.00	1.14	1.38	1.63		2.14
CPLX	0.75	0.88	1.00	1.16	1.31	1.66		2.21
PLAT			1.00	1.04	1.12	1.28		1.28
PVOL		0.92	1.00	1.09	1.17			1.27
TOOL	1.41	1.21	1.00	0.80	0.58			2.43
EXPE	1.56	1.24	1.00	0.78	0.64	0.55		2.84

distribution revealed extreme observations such as the maximum ratio, where the effort with security activities was 22.21 times higher than the effort required to develop the maintenance project. The outlier rule set an upper limit of 1.74 for this variable, which reduced the effort spent with security in the data set. Nevertheless, the new coefficient is still inside the confidence interval calculated for SECU predictor in the original version of the data set (0.692 to 1.531).

The residual quantile–quantile (Q-Q) plot for this model is very similar to the QQ plot of the Complete Data set-based model, as presented in Fig. 16.

4.2.7. Model validation

Table 9 also shows that the Residual Standard Error (RSE) for the Filtered Data set-based model is better than in the previous model. It was reduced from 0.536 to 0.364, suggesting that the predictions are more accurate.

Further results of the accuracy assessment are presented in Table 10. All accuracy metrics improved when using the Filtered Data set. The best performance of the model is shown in the 5-fold Cross-Validation approach with 64.2 percent of predictions within 25 percent of the actuals (PRED25), and 82.2 percent of the predictions within 40 percent of the actuals (PRED40).

4.2.8. The resulting model

The resulting model equation, calibrated with the 1,072 observations of the Filtered Data set and back-transformed to the original units is:

$$\widehat{\text{Effort\_h}} = 10.25 \times \text{Size\_FP}^{1.039} \times \text{SECU}^{0.707} \times \text{FAIL}^{1.386} \times \text{CPLX}^{0.920} \times \text{PLAT}^{0.563} \times \text{PVOL}^{0.607} \times \text{EXPE}^{2.413} \times \text{TOOL}^{2.166} \quad (8)$$

The resulting values for the cost drivers' multipliers are then obtained by applying the respective exponent of Eq. (8) to the initial values used in the multiple linear regression. Table 11 presents the cost drivers' values obtained. Columns VL to UH are the cost drivers' levels, Very Low (VL), Low (L), Nominal (N), High (H), Very High (VH), Extra High (XH), and Ultra High (UH). PR stands for Productivity Range, calculated as the maximum value of the cost drivers divided by its minimum value.

4.2.9. Results for the SECU cost driver

The estimation of the SECU coefficient with the Complete Data set is 1.102 with a 95% confidence interval between 0.683 and 1.521 (this is for the model without the SITE parameter). The estimation of SECU for the same model, with the Filtered Data set, was 0.707 with a confidence interval between 0.404 and 1.010 (Table 8).

In both situations, the SECU coefficient is statistically significant, with a t-value above 4 and p-value < 0.0001. Independently of considering the complete data set or removing the outliers (Filtered Data set), the null hypothesis that SECU does not influence the projects' effort can be rejected, as both estimates and confidence intervals for the SECU coefficient are safely distant from the zero value.

The resulting SECU multipliers for the 0.707 estimate of the coefficient, shown in Table 11, are lower than the initial estimates provided by experts in the Delphi sessions. Table 12 presents the multipliers for the SECU levels from both sources. According to the Filtered Data

Table 12
Comparison of SECU rating values.

Source	N	H	VH	XH	UH	PR
Expert Opinion	1.00	1.28	1.64	2.11	2.70	2.70
Filtered DS Model (coef = 0.707)	1.00	1.19	1.42	1.69	2.02	2.02
Complete DS Model (coef = 1.102)	1.00	1.31	1.73	2.28	2.99	2.99

set results, a software project planning to apply practices at the High level of the scale will spend 19% additional effort with the required security. The highest value of the SECU multiplier is 2.02 for the Ultra High level, meaning that a project will double its effort if applying the practices specified in this level.

For the Complete Data set results, the multipliers are slightly higher than the expert opinion-provided values. According to these results, the effort estimation for a project applying security practices at the high level of the SECU scale will cost 31% more than the nominal value. Projects in the extreme of the SECU scale will spend three times the effort originally estimated.

5. Discussion

The lack of knowledge about the effort required to build security in software development hinders the planning of proper resources to cope with security threats. In this study, this problem was tackled by developing a rating scale to determine the level of required software security (RQ1) and by providing effort multipliers for the levels of the scale (RQ2).

RQ1 was answered by developing a scale to identify the degree of application of security practices, classified in three groups "Security Requirements and Design", "Secure Coding and Security Tools", and "Security Verification & Validation". The resulting scale was presented in Table 1.

For RQ2, the increase in development effort caused by the growing levels of application of security practices was obtained in two ways: (1) from expert opinion as presented in Fig. 8, and (2) with the statistical model that provided the statically significant SECU cost driver effort multiplier values from the model coefficients, shown in Table 12.

5.1. Multipliers for required software security levels

The rating scale to determine levels of required security was used to collect effort data from two sources: expert opinion and project data. The productivity range provided by security experts was 2.7, from the nominal level (no security practice applied) to the highest level of the scale (practices applied rigorously and extensively). The productivity range obtained through the statistical model building was 2.02 for the Filtered Data set and 2.99 for the Complete Data set.

Previous studies reported results for the estimation of the productivity range and the multipliers for the security factor in software cost models, although not validated with observational data. Our statistical models confirm that the application of security practices during the software development process has an impact on effort, providing evidence with data from real projects.

The outcomes of the statistical models also demonstrate that the costs of secure software development are not as high as presented in previous studies. Fig. 17 shows a comparison between the prior effort models and the ones produced in this study. It can be observed that the three sets of multipliers obtained in this research (Expert Opinion, Complete Data set, and Filtered Data set) are positioned in the bottom part of the chart. Only the COCOMO II Security Extension presents a very similar result to the one obtained by the Filtered Data set-based model. The multipliers for this model were estimated in a Delphi exercise that involved commercial and aerospace organizations in 2003 (Reifer et al., 2003).

On the top part of the chart are the COSECMO min and max values. These multipliers were derived from a quotation provided by an independent agent for evaluating 5 KSLOC of infrastructure software, according to the Common Criteria Evaluation Assurance Levels (Colbert and Boehm, 2008). Another model on the top is the Weapon System, with a 1.87 multiplier for the level High of security, obtained from expert opinion with the aim of calibrating a model with 73 data points from the Korean defense domain (Lee et al., 2014). The model calibration, however, did not achieve a statistically significant coefficient for the security predictor.

The Secure OS min and max multipliers are intermediary values in this comparison. They resulted from a mini-Delphi conducted among the authors and an industry expert in an exploratory study for establishing an effort estimation model for secure operating system (OS) software development (Yang et al., 2015).

The comparison of the models suggests that the domain of the input data has an impact on the security multipliers. Recall that the data set used in this study is based on projects from bank and telecom organizations that could be classified in the *Information Systems* super-domain. Except for the COCOMO II Security Extension, whose estimates are very close to the multipliers produced by the Filtered-DS model, all other studies involved domains or applications related to the *Engineering* and *Real-Time* super-domains. This interpretation confirms previous expectations (Menzies et al., 2017) and is in line with the findings of Rosa et al. (2017), whose investigation, involving 20 agile projects from the US Department of Defense (DoD), demonstrated that the productivity in the Information Systems super-domain is higher than the productivity in the Engineering super-domain, which, in turn, is higher than the productivity in the Real-Time super-domain (Rosa et al., 2017).

Another plausible explanation for the lower values of multipliers in our results is related to the advances in automation in software development, particularly in the Security field, that tends to improve productivity. The data set analyzed in this dissertation is from projects developed over the years of 2019 and 2020 and may have benefited from security tools that were not available years ago when the estimations from the other studies were collected.

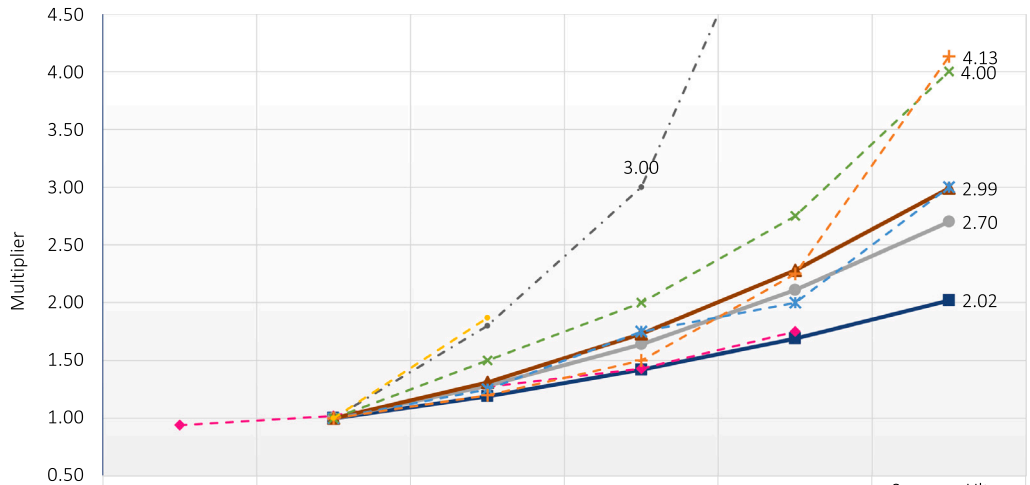
5.2. Relationship of security with other variables

Software qualities such as Security and Reliability share some characteristics – e.g. both contribute to dependability – and are commonly associated (Boehm et al., 2016; Boehm and Kukreja, 2017). This relationship has been hypothesized in some studies (Yang et al., 2015; Colbert and Boehm, 2008). However, contrary to this assumption, the collinearity test for the predictors in the analyzed data set revealed no meaningful correlation between the security predictor (SECU) and the impact of software failure (FAIL) - another denomination for Reliability. In fact, no correlation was found between security and any other predictor in the data set.

While the independence of the SECU parameter is positive for using it as a predictor in statistical model building, the low values of correlation with other variables is an interesting result per se. Studies on the interrelations of software qualities point out that there can be conflicts and synergies between qualities (Sentilles et al., 2020). For example, rigorous security tests impact positively on reliability - a synergy; on the other hand, a measure to improve security by using a single-agent key distribution system can introduce a single point of failure in terms of reliability - a conflict (Boehm and Kukreja, 2017).

5.3. Outliers for productivity and security-related effort

The identification of the outliers in the data set created a situation where the resulting coefficient for the SECU parameter can adjust the initial multipliers to a higher value, if the outliers are maintained, or adjust them to a lower value, if the outliers are removed.



	Low	Nominal	High	Very High	Extra High	Super or Ultra High
Filtered DS (this study)		1.00	1.19	1.42	1.69	2.02
COCOMO II Sec. Ext.	0.94	1.02	1.27	1.43	1.75	
Expert Opinion (this study)		1.00	1.28	1.64	2.11	2.70
Complete DS (this study)		1.00	1.31	1.73	2.28	2.99
Secure OS (min)		1.00	1.25	1.75	2.00	3.00
Secure OS (max)		1.00	1.50	2.00	2.75	4.00
COSECMO (min)		1.00	1.20	1.50	2.25	4.13
COSECMO (max)		1.00	1.80	3.00	6.00	13.50
Weapon Systems		1.00	1.87			

Fig. 17. Comparison of models' security multipliers.

The histograms for the two variables used for analyzing the outliers, productivity and security effort ratio, showed that both presented extreme values.

One example was the max value of 22.21 for the security effort ratio, indicating that the effort with security was more than 22 times the effort spent in developing the software. This specific data point is a project of 2.82 function points, developed in 14.8 h, with 329 h dedicated to security activities. An interview with the responsible for the data set informed that this is likely a case where the software was submitted to a security scanner to detect vulnerabilities for the first time and it revealed a backlog of vulnerabilities that had to be patched in order to deploy the enhancement project. Considering that this practice could have happened for other projects was well, this contextualization helps to make the case for the removal of security effort ratio outliers.

For the productivity variable, one extreme case was the data point with productivity of 680 h/PF. This project was a 4.8 function points maintenance that took 5440 h or 35.8 person-months to be developed. While the information available in the data set does not allow to state that this was a measurement error, this possibility cannot be discarded. The constant A was estimated by the Filtered data set as 10.25 h/PF, and for the Complete data set as 13.49 h/PF, what can be considered the productivity indexes if the cost drivers are ignored. Even considering the worst case scenario for all cost drivers, the productivity of 680 h/PF would not be possible.

Another aspect to consider in favor of the outliers removal was that a conservative approach was taken to determine the limits to label the outliers. The limits were calculated using a parameter that considers a probability of 95% that the sample contains no observations beyond the cutoffs (Hoaglin and Iglewicz, 1987).

5.4. Limitations

The application of the Online Delphi method allowed us to access a geographic disperse sample of security experts, recruited through LinkedIn. However, one limitation is the low representation of the sample, composed by 17 and 14 participants in the first and second round, respectively. Though good estimates in Delphi can be obtained by small groups of experts, we were not able to evaluate the real expertise of the participants, beyond their informed years of experience in Software Security. Another potential limitation is the lack of discussion among participants between the rounds. We sought to overcome that by providing a report with the quantitative results of the first round, with the distribution of the respondents' estimations, and a summary of concerns and observations provided by all the experts as an input for the second round of estimates.

Despite the large data set used in this research, the generalization of the results is limited in some aspects, such as the type and size of the projects, the software sizing method, and the variability of the cost drivers ratings.

Also, the data set is composed of maintenance projects, classified mostly as small projects. While models like COCOMO II do not differentiate development and maintenance projects when defining the cost drivers, it is not clear if larger development projects could affect the resulting multipliers.

For the sizing method, all projects applied a factor to deflate the project size for changed and deleted function points. This practice is sometimes used in industry when measuring the scope of maintenance projects that involve added, changed, and deleted functions. In a way, it normalizes the size, making it more comparable with development projects that in general contain only added functions. However, it must be considered that the use of the multipliers is conditioned to a similar approach for sizing the project to be estimated. Other sizing methods like the ISO 19761 - COSMIC Function Points are able to overcome the need to deflate the project size in maintenance projects (Dumke and Abran, 2016).

Although limited by the aspects presented above, the results of this research are nonetheless valid for the purpose of answering the research questions. Access to data sets from the software industry containing variables such as size, effort, and security information is rare, making any sample available a valuable resource.

6. Conclusion

This research aimed to develop a cost estimation model in line with current software security practices in order to quantify the effects of required software security on the software development effort.

An ordinal scale was developed, based on the sources of cost discovered through a systematic review of the literature and a survey with practitioners. The descriptions for the scale levels were evaluated and improved with the feedback obtained from estimation and security experts. Their expertise was also put in use through in-person and online Delphi sessions to collect expert estimates for the productivity range of the security cost driver.

Gathering data for building the statistical model was a great challenge in this project. After many meetings and attempts, a large data set from two organizations, containing a subset of maintenance projects that performed security activities was obtained. Based on the results of multiple linear regressions from this data set, it can be concluded that the application of software security practices can impact the cost estimations ranging from a 19% additional effort (the first level of the scale) to a 102% additional effort (the highest level of the scale).

This research builds on previous works on secure software cost models and goes one step forward by providing an empirical validation for the required software security scale. Even though the scope of the data set limits the generalization of the results, the resulting model can be used by practitioners in this field to estimate proper resources for secure software development. Additionally, the validated multipliers represent an important source of information for researchers developing investment models for software security.

Recommendations of studies to follow include further investigating the relationships between security and other cost drivers, exploring open source software repositories as a means to obtain security and productivity data, investigating the size of security features, exploring new application security standards (e.g. ISO/IEC 27034), other functional measurement standards (e.g. ISO 19761 - COSMIC Function Points), and analyzing the impact of the adoption of different levels of security on the number of vulnerabilities reported.

CRediT authorship contribution statement

Elaine Venson: Conceptualization, Data curation, Formal analysis, Project administration, Original draft, Writing – review & editing. **Bradford Clark:** Conceptualization, Data curation, Formal analysis, Original draft, Writing – review & editing. **Barry Boehm:** Supervision, Review & editing.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Data availability

The authors do not have permission to share data.

Acknowledgments

This material is based upon work supported in part by the U.S. Department of Defense through the Systems Engineering Research Center (SERC) under Contract HQ0034-19-D-0003. SERC is a federally funded University Affiliated Research Center managed by Stevens Institute of Technology.

References

Abdullah, N.A.S., Abdullah, R., Selamat, M.H., Jaafar, A., 2010. Extended function point analysis prototype with security costing estimation. In: 2010 International Symposium on Information Technology. Vol. 3, pp. 1297–1301. http://dx.doi.org/10.1109/ITSIM.2010.5561460.

Abu-Taieh, E.M.O., 2017. Cyber security body of knowledge. In: 2017 IEEE 7th International Symposium on Cloud and Service Computing (SC2). pp. 104–111. http://dx.doi.org/10.1109/SC2.2017.23.

Amoroso, E., 2018. Recent progress in software security. IEEE Softw. 35 (2), 11–13. http://dx.doi.org/10.1109/MS.2018.1661316.

Beckers, K., Faßbender, S., Hatebur, D., Heisel, M., Côté, I., 2013. Common criteria compliant software development (CC-CASD). In: Proceedings of the 28th Annual ACM Symposium on Applied Computing - SAC '13. ACM Press, Coimbra, Portugal, p. 1298. http://dx.doi.org/10.1145/2480362.2480604, URL http://dl.acm.org/citation.cfm?doid=2480362.2480604.

Bedi, P., Gandotra, V., Singhal, A., Narang, H., Sharma, S., 2013. Mitigating multi-threats optimally in proactive threat management. SIGSOFT Softw. Eng. Notes 38 (1), 1–7. http://dx.doi.org/10.1145/2413038.2413041, URL http://doi.acm.org/10.1145/2413038.2413041.

Boateng, G.O., Neilands, T.B., Frongillo, E.A., Melgar-Quiñonez, H.R., Young, S.L., 2018. Best practices for developing and validating scales for health, social, and behavioral research: A primer. Front. Public Health 6, http://dx.doi.org/10.3389/fpubh.2018.00149, URL https://www.ncbi.nlm.nih.gov/pmc/articles/PMC6004510/, tex.pmcid: PMC6004510.

Boehm, B.W., 1981. Software Engineering Economics, first ed. Prentice Hall, Englewood Cliffs, N.J.

Boehm, B.W., Abts, C., Brown, A.W., Chulani, S., Clark, B.K., Horowitz, E., Madachy, R., Reifer, D.J., Steece, B., 2000. Software Cost Estimation with COCOMO II, first ed. Prentice Hall Press, Upper Saddle River, NJ, USA.

Boehm, B., Chen, C., Srisopha, K., Shi, L., 2016. The key roles of maintainability in an ontology for system qualities. In: INCOSE International Symposium. Vol. 26, (1), pp. 2026–2040. http://dx.doi.org/10.1002/j.2334-5837.2016.00278.x, URL https://onlinelibrary.wiley.com/doi/abs/10.1002/j.2334-5837.2016.00278.x, tex.copyright: Copyright © 2016 Barry Boehm. Published and used by INCOSE with permission.

Boehm, B., Kukreja, N., 2017. An initial ontology for system qualities. Insight (American Society of Ophthalmic Registered Nurses) 20 (3), 18–28. http://dx.doi.org/10.1002/inst.12160, URL https://onlinelibrary.wiley.com/doi/abs/10.1002/inst.12160.

Böhme, R., 2010. Security metrics and security investment models. In: Echizen, I., Kunihiro, N., Sasaki, R. (Eds.), Advances in Information and Computer Security. Vol. 6434, Springer Berlin Heidelberg, Berlin, Heidelberg, pp. 10–24. http://dx.doi.org/10.1007/978-3-642-16825-3_2.

Chehrazi, G., Heimbach, I., Hinz, O., 2016. The impact of security by design on the success of open source software. In: ECIS 2016 Proceedings. p. 18, URL http://aisel.aisnet.org/ecis2016_rp/179.

Chiesa, R., De Luca Saggese, M., 2016. Data breaches, data leaks, web defacements: Why secure coding is important. In: Ciancarini, P., Sillitti, A., Succi, G., Messina, A. (Eds.), Proceedings of 4th International Conference in Software Engineering for Defence Applications. Vol. 422, Springer International Publishing, Cham, pp. 261–271. http://dx.doi.org/10.1007/978-3-319-27896-4_22, URL http://link.springer.com/10.1007/978-3-319-27896-4_22.

Chulani, S., 2001. Bayesian analysis of software cost and quality models. In: Proceedings IEEE International Conference on Software Maintenance. ICSM 2001. pp. 565–568. http://dx.doi.org/10.1109/ICSM.2001.972773.

Colbert, E., Boehm, D.B., 2008. Cost estimation for secure software & systems. In: ISPA/SCEA 2008 Joint International Conference. The Netherlands, p. 9.

Common Criteria, 2017a. Common criteria for information technology security evaluation v3.1 - part 1: Introduction and general model.

Common Criteria, 2017b. Common criteria for information technology security evaluation v3.1 - part 3: Security assurance components.

DeVellis, R.F., 2011. Scale Development: Theory and Applications, third ed. SAGE Publications, Inc, Thousand Oaks, Calif.

Donohoe, H., Stellefson, M., Tennant, B., 2012. Advantages and limitations of the e-Delphi technique. Am. J. Health Educ. 43 (1), 38–46. http://dx.doi.org/10.1080/19325037.2012.10599216.

Dumke, R., Abran, A., 2016. COSMIC Function Points: Theory and Advanced Practices. CRC Press, Google-Books-ID: 1fFdhxO8mnIC.

Duncan, B., Whittington, M., 2014. Compliance with standards, assurance and audit: Does this equal security? In: Proceedings of the 7th International Conference on Security of Information and Networks. SIN'14, ACM, New York, NY, USA, pp. 77:77–77:84. http://dx.doi.org/10.1145/2659651.2659711, URL http://doi.acm.org/10.1145/2659651.2659711.

Geer, D., 2010. Are companies actually using secure development life cycles? Computer 43 (6), 12–16. http://dx.doi.org/10.1109/MC.2010.159.

Georg Schmitt, 2018. High-level cybersecurity meeting warns of dire effects of cyberattacks on prosperity, innovation and global collaboration. World Economic Forum URL https://www.weforum.org/press/2018/11/high-level-cybersecurity-meeting-warns-of-dire-effects-of-cyberattacks-on-prosperity-innovation-and-global-collaboration/.

Hein, D., Saiedian, H., 2009. Secure software engineering: Learning from the past to address future challenges. Inf. Secur. J.: Glob. Perspect. 18 (1), 8–25. http://dx.doi.org/10.1080/19393550802623206.

Heitzenrater, C., Bohme, R., Simpson, A., 2016. The days before zero day: Investment models for secure software engineering. p. 14.

Heitzenrater, C., Simpson, A., 2016a. A case for the economics of secure software development. In: Proceedings of the 2016 New Security Paradigms Workshop. NSPW'16, ACM, New York, NY, USA, pp. 92–105. http://dx.doi.org/10.1145/3011883.3011884, URL http://doi.acm.org/10.1145/3011883.3011884.

Heitzenrater, C., Simpson, A., 2016b. Misuse, abuse and reuse: Economic utility functions for characterising security requirements. In: 2016 11th International Conference on Availability, Reliability and Security (ARES). pp. 572–581. http://dx.doi.org/10.1109/ARES.2016.90.

Heitzenrater, C., Simpson, A., 2016c. Software security investment: The right amount of a good thing. In: 2016 IEEE Cybersecurity Development (SecDev). pp. 53–59. http://dx.doi.org/10.1109/SecDev.2016.020.

Hoaglin, D.C., Iglewicz, B., 1987. Fine-tuning some resistant rules for outlier labeling. J. Amer. Statist. Assoc. 82 (400), 1147–1149. http://dx.doi.org/10.2307/2289392, URL http://www.jstor.org/stable/2289392.

Howard, M., Lipner, S., 2006. The Security Development Lifecycle. Microsoft Press, Redmond, WA, USA.

Kaluvuri, S.P., Bezzi, M., Roudier, Y., 2014. A quantitative analysis of common criteria certification practice. In: Eckert, C., Katsikas, S.K., Pernul, G. (Eds.), Trust, Privacy, and Security in Digital Business. vol. 8647, Springer International Publishing, Cham, pp. 132–143. http://dx.doi.org/10.1007/978-3-319-09770-1_12, URL http://link.springer.com/10.1007/978-3-319-09770-1_12.

Khodyakov, D., Grant, S., Denger, B., Kinnett, K., Martin, A., Peay, H., Coulter, I., 2020. Practical considerations in using online modified-delphi approaches to engage patients and other stakeholders in clinical practice guideline development. The Patient - Patient-Centered Outcomes Res. 13 (1), 11–21. http://dx.doi.org/10.1007/s40271-019-00389-4.

Kott, A., Swami, A., McDaniel, P., 2014. Security outlook: Six cyber game changers for the next 15 years. Computer 47 (12), 104–106. http://dx.doi.org/10.1109/MC.2014.366.

Kuhn, R., Raunak, M., Kacker, R., 2017. It doesn't have to be like this: Cybersecurity vulnerability trends. IT Prof. 19 (6), 66–70. http://dx.doi.org/10.1109/MITP.2017.4241462.

Lee, T., Gu, T., Baik, J., 2014. MND-SCEMP: an empirical study of a software cost estimation modeling process in the defense domain. Empir. Softw. Eng. 19 (1), 213–240. http://dx.doi.org/10.1007/s10664-012-9220-1, URL https://link.springer.com/article/10.1007/s10664-012-9220-1.

Lee, M.-g., Sohn, H.-j., Seong, B.-m., Kim, J.-b., 2016. Secure software development lifecycle which supplements security weakness for CC certification. Int. Inf. Inst. (Tokyo). Inf.; Koganei 19 (1), 297–302, URL http://search.proquest.com/docview/1776684205/abstract/3E850391C94D4932PQ/1, tex.copyright: Copyright International Information Institute Jan 2016.

McGraw, G., 2006. Software Security: Building Security, first ed. Addison-Wesley Professional, Upper Saddle River, NJ.

McGraw, G., 2013. Cyber war is inevitable (unless we build security in). J. Strateg. Stud. 36 (1), 109–119. http://dx.doi.org/10.1080/01402390.2012.742013.

Menzies, T., Yang, Y., Mathew, G., Boehm, B., Hihn, J., 2017. Negative results for software effort estimation. Empir. Softw. Eng. 22 (5), 2658–2683. http://dx.doi.org/10.1007/s10664-016-9472-2, URL https://link.springer.com/article/10.1007/s10664-016-9472-2.

Migues, S., Steven, J., Ware, M., 2019. Building security in maturity model (BSIMM) - Version 10. Technical Report 10, Synopsys Software Integrity Group, p. 92, URL https://www.bsimm.com/download.html.

Morrison, P., Smith, B.H., Williams, L., 2017. Surveying security practice adherence in software development. In: Proceedings of the Hot Topics in Science of Security: Symposium and Bootcamp. In: HoTSoS, ACM, New York, NY, USA, pp. 85–94. http://dx.doi.org/10.1145/3055305.3055312, URL http://doi.acm.org/10.1145/3055305.3055312.

Morton, K.L., Atkin, A.J., Corder, K., Suhrcke, M., Turner, D., van Sluijs, E.M.F., 2017. Engaging stakeholders and target groups in prioritising a public health intervention: the creating active school environments (CASE) online Delphi study. BMJ Open 7 (1), e013340. http://dx.doi.org/10.1136/bmjopen-2016-013340, URL http://bmjopen.bmj.com/lookup/doi/10.1136/bmjopen-2016-013340.

Olama, M.M., Nutaro, J., 2013. Secure it now or secure it later: the benefits of addressing cyber-security from the outset. In: Cyber Sensing 2013. Vol. 8757, International Society for Optics and Photonics, p. 87570L. http://dx.doi.org/10.1117/12.2015465, URL https://www-spiedigitallibrary-org.libproxy1.usc.edu/conference-proceedings-of-spie/8757/87570L/Secure-it-now-or-secure-it-later--the-benefits/10.1117/12.2015465.short.

OWASP SAMM Project, 2017. Software Assurance Maturity Model (SAMM): A guide to building security into software development - v1.5. Technical Report Version 1.5, p. 72, URL https://owaspsamm.org/.

Peeters, J., Dyson, P., 2007. Cost-effective security. IEEE Secur. Priv. 5 (3), 85–87. http://dx.doi.org/10.1109/MSP.2007.56.

Reifer, D.J., Boehm, B.W., Gangadharan, M., 2003. Estimating the cost of security for COTS software. In: COTS-Based Software Systems. In: Lecture Notes in Computer Science, Springer, Berlin, Heidelberg, pp. 178–186. http://dx.doi.org/10.1007/3-540-36465-X_17, URL https://link.springer.com/chapter/10.1007/3-540-36465-X_17.

Rosa, W., Madachy, R., Clark, B., Boehm, B., 2017. Early phase cost models for agile software processes in the US DoD. In: 2017 ACM/IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM). pp. 30–37. http://dx.doi.org/10.1109/ESEM.2017.10.

SAFECode, 2018. Fundamental practices for secure software development: Essential elements of a secure development lifecycle program. URL https://safecode.org/wp-content/uploads/2018/03/SAFECode_Fundamental_Practices_for_Secure_Software_Development_March_2018.pdf.

Sentilles, S., Boehm, B., Trubiani, C., Franch, X., Koziole, A., 2020. Software Qualities and their Dependencies Report on two editions of the workshop. ACM SIGSOFT Softw. Eng. Notes 45 (1), 31–33. http://dx.doi.org/10.1145/3375572.3375581.

Tierney, J., Boswell, T., 2017. Common criteria: Origins and overview. In: Mayes, K., Markantonakis, K. (Eds.), Smart Cards, Tokens, Security and Applications. Springer International Publishing, Cham, pp. 193–216. http://dx.doi.org/10.1007/978-3-319-50500-8_8.

Tukey, J.W., 1977. Exploratory Data Analysis. In: Addison-Wesley Series in Behavioral Science, Addison-Wesley PubCo, Reading, Mass.

Venson, E., Alfayez, R., Marília M. F., G., Rejane M. C., F., Boehm, B., 2019a. The impact of software security practices on development effort: An initial survey. In: 2019 ACM/IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM). pp. 1–12. http://dx.doi.org/10.1109/ESEM.2019.8870153.

Venson, E., Guo, X., Yan, Z., Boehm, B., 2019b. Costing secure software development: A systematic mapping study. In: Proceedings of the 14th International Conference on Availability, Reliability and Security. ARES '19, ACM, New York, NY, USA, pp. 9:1–9:11. http://dx.doi.org/10.1145/3339252.3339263, URL http://doi.acm.org/10.1145/3339252.3339263.

Verendel, V., 2009. Quantified security is a weak hypothesis: A critical survey of results and assumptions. In: Proceedings of the 2009 Workshop on New Security Paradigms Workshop. NSPW '09, ACM, New York, NY, USA, pp. 37–50. http://dx.doi.org/10.1145/1719030.1719036, URL http://doi.acm.org/10.1145/1719030.1719036.

Williams, L., McGraw, G., Migues, S., 2018. Engineering security vulnerability prevention, detection, and response. IEEE Softw. 35 (5), 76–80. http://dx.doi.org/10.1109/MS.2018.290110854.

Yang, Y., Du, J., Wang, Q., 2015. Shaping the effort of developing secure software. Procedia Comput. Sci. 44 (Supplement C), 609–618. http://dx.doi.org/10.1016/j.procs.2015.03.041, URL http://www.sciencedirect.com/science/article/pii/S187705091500277X.

Elaine Venson received the Ph.D. degree in computer science from the University of Southern California in 2021. She is currently working as assistant professor for the bachelor's degree program in Software Engineering at the University of Brasília (UnB). Her research interests include software security, software cost estimation, and software process improvement.

Bradford K. Clark received the Ph.D. degree in computer science from the University of Southern California in 1997. He is currently the vice-president of Software Metrics Inc. – a Virginia based consulting company. His research interests include software cost and schedule data collection, analysis, and modeling. He has co-authored a book with Barry Boehm and others on Software Cost Estimation with COCOMO II. He is a former US Navy A-6 Intruder pilot.

Barry W. Boehm (Member, IEEE) received the BA degree from Harvard in 1957, the MS and Ph.D. degrees in mathematics from UCLA in 1961 and 1964, respectively. He was a distinguished professor of software engineering with the Computer Science Department, University of Southern California, and known for his many contributions to software engineering. He was the recipient of an honorary ScD in computer science from the University of Massachusetts and in software engineering from the Chinese Academy of Sciences in 2011. Dr. Barry W. Boehm passed away in August 20, 2022.