The Days Before Zero Day: Investment Models for Secure Software Engineering

Chad Heitzenrater*‡
chad.heitzenrater@cs.ox.ac.uk
*U.S. Air Force Research Laboratory
Information Directorate
525 Brooks Road
Rome NY 13441, USA

Rainer Böhme†
rainer.boehme@uibk.ac.at
†*Security and Privacy Laboratory*
Universität Innsbruck
Technikerstraße 21A
6020 Innsbruck, Austria

Andrew Simpson‡
andrew.simpson@cs.ox.ac.uk
‡*Department of Computer Science*
University of Oxford
Wolfson Building, Parks Road
Oxford OX1 3QD, UK

***Abstract*—While the majority of security practice — and spending — is focused on post-development products and enterprise approaches, some have sought to change the focus of security from the networks we manage to the systems we build. The burgeoning Secure Software Engineering (SSE) community has sought to identify and espouse activities, built upon traditional software engineering, that address the introduction of vulnerabilities as a means of stemming the growing tide of security problems before they can be realised. It is widely believed that not only do such approaches hold promise to limit exposure and reduce security incidents, but they are also a valid security investment that decreases overall security expenditure. While many initiatives are now underway to codify such SSE practices, a treatment of the economic considerations has yet to be conducted. We propose an initial model that captures SSE investment as a means of reducing defender uncertainty regarding vulnerabilities, while raising the cost to the attacker. This approach is instantiated as a companion process to traditional security models, and we use the Iterated Weakest Link (IWL) model of (post-deployment) security investment to demonstrate how defender security investment can be optimised over the system's lifecycle. The results indicate both an increased return on security investment — the Return on Secure Software Process (ROSSP) — as well as reduced post-deployment costs. It is our hope that this model paves the way for a more comprehensive treatment of security investment that unifies pre- and post-security investment, leading to a more comprehensive view of security in software systems.**

1. Introduction

Computer security has become synonymous with security products. As consumers, we install anti-virus solutions, keep our applications patched and up to date, and ensure the configuration of our firewalls and routers. Businesses invest in a range of vendor products and services, often creating a heterogeneous landscape of partial solutions. The security industry itself can be characterised as a series of 'next' solutions, with security professionals trained on a succession of security appliances. Security investment has become a reactionary practice — optimising the expenditure of resources to fix problems that are largely preventable, leading some to characterise the current environment as the "dark ages" of security thinking [1].

That is not to say that system-level, preventative approaches aren't a necessary component to security. In practice, disparate systems must come together to form the enterprise, and no amount of planning can predict the emergent security behaviours. However, exclusive focus on enterprise security investment is an incomplete approach that fails to reflect the sources of vulnerabilities [2] and the inherent fallibility of discovery approaches [3]. Applications are increasingly a "prime vector into an organisation" for someone looking to bypass perimeter defences [4], leading to a growing conviction that the correct context in which to address computer security failures is within development [5]. Yet models for information security investment have not yet caught up to this view, with a continued focus on post-development optimisation of reactionary approaches.

In this paper, we introduce a model for security investment that incorporates the software development process. Secure Software Engineering (SSE) codifies activities widely seen as the best redress of vulnerability introduction, but these have received little attention from the information security investment community. Our model provides mechanisms for practitioners to understand SSE investments across a system's life cycle, in order to maximise security gains.

Following an overview of relevant software engineering and security investment concepts in Section 2, we define the problem addressed by SSE investment modelling in Section 3. Next, we provide an overview of the Iterated Weakest Link (IWL) security investment model [6], serving as the basis upon which we form our initial model instantiation. Our model for secure software engineering investment, IWL-SSE, is presented in Section 4 along with a discussion of the model's operation. An analysis of the broader implications is presented in Section 5, where we introduce the metric of Return on Secure Software Process (ROSSP) as a mechanism for comparative SSE decision-making. We conclude with thoughts on future directions in Section 6.

Approved for Public Release; Distribution Unlimited: 88ABW-2016-0892 20160229

2. Background

Of the many accepted security tenets, one commonly cited during project planning is that "Security is a process, not a product" [7]. Despite this, a treatment of the processes involved in system development are often taken for granted or ignored by the security investment community. Our model is built on the premise that investment into security practices during system development is critical to the production of truly secure systems. This section provides the necessary background to further develop the argument to support this belief.

2.1. Software Engineering Process

Fundamental to system and software development is the concept of a System Development Lifecycle, or SDLC. While the term is often used generally — with 'S' sometimes meaning 'Software', rather than 'System' — the US National Institute for Standards and Technology (NIST) sought to codify the concept within [8], specifying security practices for five phases of a system's lifetime: Initiation, Development/Acquisition, Implementation/Assessment, Operations/Maintenance, and Disposal. Many of the common security concerns, such as accreditation to standards, configuration management and patching, and perimeter security pertain only to the Implementation/Assessment and Operations/Maintenance stages.

It is within the context of an SDLC that an organisation defines their software development process, potentially spanning each phase but often concentrated in the Initiation and Development/Acquisition phases. Such process models are one of the most fundamental aspects of software development, governing the inclusion, frequency, timing and scope of development activities (requirements, architecture, design, coding, testing, and release). Common variants of software processes include the waterfall, evolutionary, iterative/incremental, spiral, prototyping, and agile models; for a primer, we refer the reader to [9]. Consideration of a process model requires taking into account organisational and project constraints, which range from the consideration of existing systems, scope of the project (standalone versus part of a product line family), and the constitution of the project team.

In addition to providing structure for the management of software developments, software engineering has long sought to quantify the economics behind software development. It has long been known that investment decisions made early in the software process are more cost-effective [10], with empirical studies demonstrating the escalation of cost over the system's lifetime (Table 1). Such data is often the basis for investment decisions throughout the development process; however, with few good metrics for security such quantified approaches have yet to provide the same insight into security investment.

Phase	Software cost factors
Requirements	$1\times$
Design	$5\times - 7\times$
Build	$10\times - 26\times$
Test	$50\times - 177\times$
Operations	$100\times - 1000\times$

TABLE 1: Software cost estimates for remediation in various phases (from [11]).

2.2. Secure Software Engineering

While the practice of software engineering is generally concerned with the development of quality software, the burgeoning community of secure software engineering seeks to augment specific process steps with security-oriented exercises. To this end, it has been said that "preventing the introduction of vulnerabilities prior to release, rather than patching vulnerabilities afterwards is THE challenge SSE rises to meet" [2]. This line of reasoning draws a distinction between *application security* — largely focused on securing software after it has been written, and more closely related to the network-centric approach to security — and *software security*, which seeks to "leverage good software engineering practice and involve thinking about security early in the software lifecycle" [12].

These practices are focused on the identification and removal of *flaws*, or errors in design, and *bugs*, which are errors in implementation [13]. Prevailing secure software engineering wisdom is that vulnerability sources are 50% the result of flaws, and 50% the result of bugs [14]. To date, efforts in software security toward these ends can be characterised as having two directions:

- *Development processes, primarily focused on the use of process and tools to reduce implementation errors*. Practices include OWASP CLASP[1], Microsoft SDL[2], and Adobe SPLC[3]. The most broadly applicable (and most cited) of these is perhaps the SSDL-Touchpoints [5], which prescribes seven practices that cover requirements to operations.
- *Practice recommendations, aimed at addressing common mistakes within a current development process*. These include prescriptive practices, such as the IEEE CSD Top 10 Architectural Flaws [15] and OWASP Top 10 [16], as well as broad meta-models such as Building Security In Meta-Model (BSIMM) [17] and OWASP OpenSAMM [18] that seek to catalogue widely practiced activities. BSIMM is a study of existing software security initiatives within different organisations, which aims to inform the wider software security community. It

1. Open Web Application Security Project Comprehensive, Lightweight Application Security Process: www.owasp.org/index.php/Category:OWASP_CLASP_Project

2. Microsoft Secure Development Lifecycle: www.microsoft.com/en-us/sdl/

3. Adobe Secure Product Life Cycle: www.adobe.com/security/proactive-efforts.html

is heralded as "a reflection of the current state of Software Security" [17], identifying practices such as abuse case and security-specific requirement definition (at the requirements phase), risk analysis (at the requirements, architecture and design, and testing phases), risk-based security tests (at test planning), code review (at the code phase), penetration testing (at testing and fielding), and security operations (following fielding).

These categories are not fully distinct; for instance, BSIMM includes Touchpoints as one of the four recommended practice categories (along with governance, intelligence, and deployment activities). In addition, they share a common limitation; while prescriptive on the various steps to take, little is specified as to the amount of effort to dedicate towards these practices.

2.3. Information Security Investment

In the field of information security investment, models tend to be focused on the enterprise level. These models examine trade-offs between investments in products rather than considering the processes associated with the development and deployment of systems. Such focus limits security investment trade-off considerations within a single phase of the software's life, ignoring its creation and post-deployment existence. When the implications of security decisions fail to consider the software lifecycle, risks are ignored, options are limited, and the resulting systems having enlarged attack surfaces and inherent complexity at their interfaces [19].

Our model is best placed in context with other models that establish optimal security investment in various contexts, using tools such as multi-objective optimisation and game-theoretic relationships; a review of such models can be found in [20]. Theoretical underpinnings of such investment models are traceable to [21]. Example game-theoretic models include those presented in [22], [23] and [24]. Issues with the application of investment metrics to security investment are discussed in [25] and [26]. Recent game-theoretic models that seek to examine these trade-offs, such as those in [27], present a natural intersection between the development-level aspects of our model and the higher-level systems security thinking required for enterprise-level consideration. Others have sought to push such considerations to earlier in the process by defining mechanisms that can best be characterised as being pre-deployment, system-level processes. Concepts such as the Security Attribute Evaluation Method (SAEM) [28] and the Appropriate and Effective Guidance for Information Security (AEGIS) [29] consider system security at the system level, but still as a post-development step. Generally risk-based, these methods rely on the existence of system artefacts (designs, components, etc.) to focus asset-based decisions — largely in the absence of cost considerations.

The challenge of combining considerations of scope, cost and timing has proven a limitation in the research to date regarding security investment models, and is indicative of the broader lack of a comprehensive treatment of security investment spanning the life of a software-based system. We argue that it is essential to provide project managers with appropriate tools to make rational decisions regarding their investment into secure software process elements. The motivation for this paper is the development of tools that combine elements of software engineering, secure software process, and information security economics to provide usable tools to this end.

3. Problem Statement

Between current practice and the ideal promised by SSE, a logical question arises: Is there an allocation of project resources that provides a more efficient outcome?

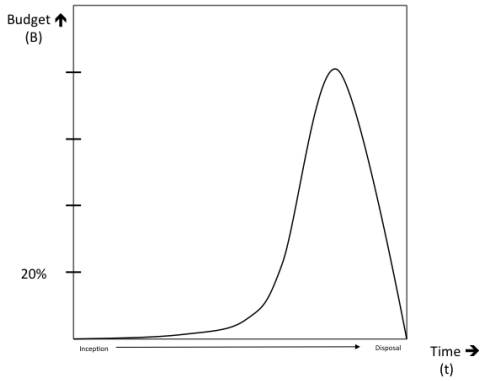
3.1. Software Process Investment

The security investment of a software system can be characterised as a series of decisions spread over the SDLC phases, starting with Inception and ending with the system's Disposal. We can then conceive of a system as having a fixed budget B, with some allocation of B dedicated to security at any point in time t (denoted B_s). Projecting this into a Cartesian space, we find that B_s may take on a variety of forms; Figures 1a – 1d present conceptualisations of how such security investment might occur.

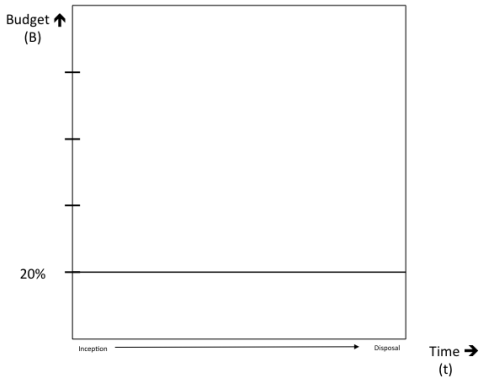
Figure 1a presents what might arguably be considered the current state: a very low security investment effort over the early stages of the process, quickly increasing to consume an ever-increasing share of the project costs. In addition to questions of efficiency in such an investment approach, such a profile allocates resources to security to the potential detriment of other investment opportunities (e.g. new functionality, or code maintenance) in later phases. Figure 1b, while more controlled, might not depict a more favourable allocation, as it would imply that security investment in any given phase is equally beneficial. Many might consider Figure 1c to be the most likely scenario to result in effective investment, with security investment ramping up as the security concerns — and system artefacts — become more tangible. However, the SSE community might argue that an investment profile such as that illustrated by Figure 1d is the most appropriate. This reflects a belief that up-front investment is the most effective means to achieving security, reducing the need for later investment into activities such as accreditation, patching, and breach remediation. The question of which of these depictions, if any, reflects a more efficient reality is the focus of this research.

3.2. Assumptions and Constraints

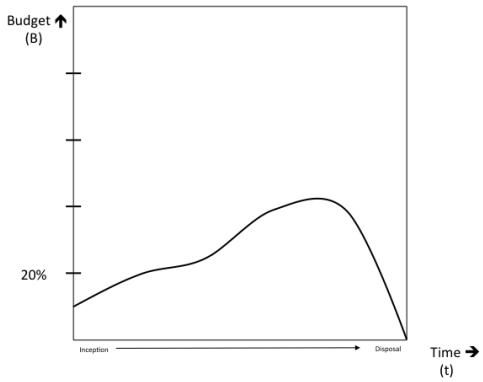
Our consideration of the problem space is focused on the application of SSE principles to a specific development environment, modelled on software engineering process considerations and followed by a weakest-link model of attacker progression. We seek to address the investment into SSE within the following context.



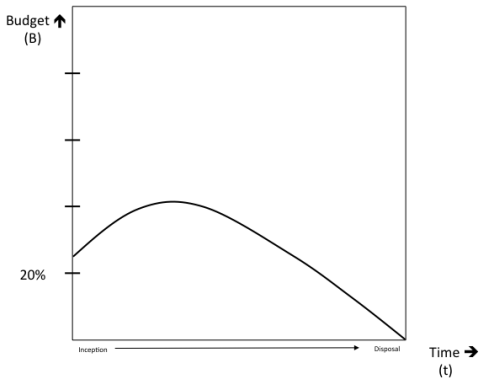
(a) Notional depiction showing the bulk of security investment occurring late in the lifecycle (e.g. post-deployment).



(b) Notional depiction of a constant security budget B_s as a fixed proportion of the per-phase budget B (here, as 20%) over the course of a project.



(c) Notional depiction of security investment that starts early in the lifecycle and slowly increases.



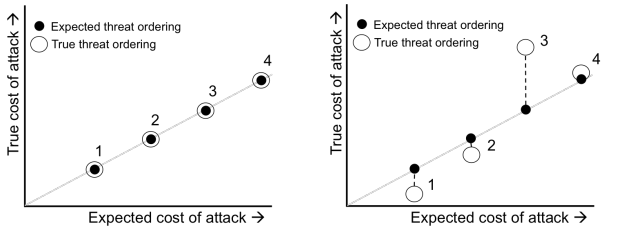
(d) Notional depiction of security investment at the early stages, leading to reduced investment in later phases.

Figure 1: Depictions of security investment over the System Development Lifecycle (SDLC)

1) We assume standalone, greenfield development, where all security investment decisions are left to the programme manager, and the implications are confined to the individual system. This removes any concern over interfaces, integration, or existing frameworks altering investment decisions.
2) We assume a sequential development lifecycle with distinguishable phases, where the development process is well-defined, controlled, and guided by a programme manager. This is easily thought of as a waterfall or incremental model, with defined Architecture/Design and Code/Test phases. While only two phases are considered here, extension to broader aspects of the software process is discussed in Section 6. In addition to simplifying the model, there is evidence that such models remain a dominant approach in many development environments [30], [31].
3) We assume that the developer is involved in the fielding and operation of the development, or that this information is reliably and truthfully conveyed. The conveyance of information is directly related to the uncertainty in the process artefacts upon which security decisions are based, and is a key element to our model. This is perhaps one of the most straightforward software engineering contexts to conceive, yet still would not be uncommon in practice. By employing such a context, we seek to define our model in the broadest sense, while making it recognisable and easy to adopt and adapt by practitioners.
4) We assume that: once a flaw or bug is found, it is fixed; the fix enacted is correct; and repeated iterations of SSE activities occur with the same effectiveness each time. While there are indications that this is often not the case [32], the impact of

such process complexities and secondary effects are left to future investigation.

We focus on the practices within the SDDL-Touchpoints domain of the BSIMM: Architecture Analysis (AA), Code Review (CR), and Security Testing (ST). Each of these activities has long been a part of established software development practice, with SDDL-Touchpoints and BSIMM supplying a security focus that augments common practice. However, little to no guidance is provided as to the magnitude of resource investment that leads to the successful completion of any of these steps, or as a collective.



(a) Attack profile with certainty ($\sigma = 0$).

(b) Attack profile with uncertainty ($\sigma > 0$).

Figure 2: Attack profile under certainty (left) and uncertainty (right).

4. Model

In order to appraise the development of secure systems in their entirety, we examine how the SSE process can be represented within the general class of information security economic models. The goal of this work is not to ignore or replace the existing class of models, whose structure and assumptions capture vital aspects of various security scenarios, but rather to augment their construction. For this initial examination, we have chosen to utilise the *Iterated Weakest Link* (IWL) model [6] as the modelling construct we will expand through the addition of SSE considerations. IWL supplies a number of features that make it an ideal candidate for such supplementing constructs, as it focuses on optimisation of defender decisions based on starting conditions that lie at the heart of software engineering: quality (in the form of cost to attacker) and uncertainty regarding vulnerabilities. SSE practices exhibit indications of weakest link behaviour as well, as reflected in the motivation for the IWL: "The most careless programmer in a software firm can introduce a critical vulnerability" [6]. Common practices such as 'top 10 list'–driven reviews and the use of code analysis tools and rulesets evolve with the realisation of 'yesterday's attack'; they are intended to place focus on the most common errors, with increasingly deeper review and analysis driving both security and cost.

A review of the IWL is now presented in order to establish the conditions required for our model of SSE investment.

4.1. Overview of the Iterated Weakest Link Model

Central to the IWL is the premise that attacks are "unknowable and hence innumerable in advance" [33]. This leads to the definition of a model that emphasises dynamic, adaptive investments over time. The authors attempt to capture the iterative nature of security investment through a focus on the following three key characteristics.

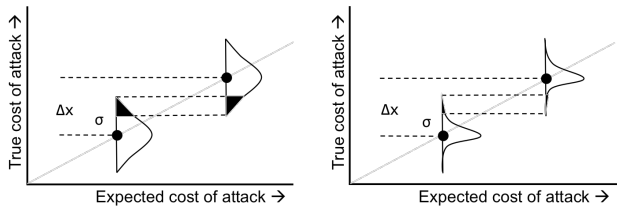
- Defender uncertainty regarding which components are weakest is a key consideration in investment decisions.
- Defence plays out as an iterative process of attacking and defending successive weakest links.
- Countermeasures can be represented as interdependent (rendering the diminishing marginal return of information security investment endogenous in this model).

The model demonstrates conditions where under-investment in security can be a rational action, given that: a) reactive investment is possible; b) uncertainty exists about the attacker's relative capability to exploit different threats; c) successful attacks are not catastrophic; and d) the sunk cost to upgrade the defence configuration is relatively small [6]. This model is applied to phishing in online crime and payment card security, with subsequent papers [33] examining the use of penetration testing as a means to conduct such uncertainty reduction.

The IWL seeks to model the protection of a set of assets a, from which a return of r per-period is enjoyed by the defender. This is jeopardised by n possible components threatened by attack (e.g. attack vectors), each with an associated cost of attack. In IWL, each successive threat $(1, \ \ldots, \ n)$ has an increase in cost of Δx over the previous threat, forming an attack gradient. In the original IWL, this was set at $\Delta x = 1$, specifying that each defender investment increased the cost to the attacker linearly.

While the true ordering of n is presumed to be known by the attacker (x), this information is unknown to the defender who must form an expected ordering of likely threats (e.g. through attack modelling as specified in BSIMM's Intelligence domain [17]). The defender's ordering is denoted $\overline{x}$ (where $\overline{x}_1 \leq \overline{x}_i \leq \overline{x}_n$). If the defender's attack modelling is perfect, they will generate an attack profile that matches the attacker's profile; this is the case depicted in Figure 2a. In this instance we can say the defender (dots) has absolute certainty regarding the attacks (rings). However, a much more likely scenario is one where the defender's estimate does not fully align with the attacker's profile, leading to an incorrect ordering. Such a case is illustrated in Figure 2b, where defender uncertainty leads to an incorrect ordering (i.e. threats 3 and 4), resulting in misplaced investment by the defender. This potential for a misalignment of threats and defences by the defender is captured by the IWL as σ, specifying the degree of the defender's uncertainty.

Defender investment in IWL plays out over discrete time $t = (1, \ \ldots, \ t_{\max})$. At each t, the defender forms a defensive configuration represented by a vector $\mathbf{d}_t$. The elements d_i of $\mathbf{d}_t \in \{0, 1\}^n$ indicate that defence against the i-th threat



(a) Defender attack ordering with high uncertainty, potentially leading to incorrect ordering of attack vectors.

(b) Defender attack ordering with low uncertainty, reducing the potential for incorrect attack vector ordering.

Figure 3: The relationship between uncertainty σ and attack gradient Δx in the establishment of defender attack order.

is implemented ($d_i = 1$) or not ($d_i = 0$). The summation of these defences is represented by k, guarded at a unit cost (1) per protection, per round.

The knowledge employed by the defender to make investment decisions follows [6], with the expected costs for the i-th threat represented as:

$$\bar{x}_i = \bar{x}_1 + (i-1) \cdot \Delta x$$

with

$$\Delta x > 0 \tag{1}$$

The unknown true costs are modelled as a Gaussian random variable $\mathcal{N}$, with mean x_i and standard deviation $\sigma/\Delta x$ (censored to values $x_i > 0$):

$$x_i = \sup(0, \chi_i)$$

with

$$\chi_i \sim \mathcal{N}(\bar{x}_i, \sigma/\Delta x) \tag{2}$$

The role of σ in the IWL is visualised in Figure 3: a higher value of σ leads to a wider distribution, increasing the probability that the defender's ordering is incorrect and results in misplaced security investment (Figure 3a). However, with lower uncertainty (i.e. a smaller standard deviation) this probability is decreased (Figure 3b).

The value of the attack gradient (Δx) also contributes to the overall security investment decision: the larger Δx, the less overlap in expected threat costs for a given σ. This is depicted in Figures 2 and 3 as the slope of the line on which the actual attack costs lie relative to the distribution of defender expectations. Therefore, the effect of these parameters on the security investment is best thought of as a ratio, $\sigma/\Delta x$ (representing an increase in attacker cost for a given amount of uncertainty). We make this distinction as an extension of the exposition in [6], as the original IWL employed a unit cost for Δx rendering this distinction moot ($\sigma/\Delta x = \sigma/1 = \sigma$).

The model operation starts with the defender specifying an initial k at $t = 0$. In each subsequent round ($t = 1, \ \ldots, \ t_{\text{max}}$) the attacker may choose to exploit the component with the least true cost not covered by $\mathbf{d}_t$, looting a fraction z from asset a — but only if the benefits exceed the cost to attack. In the face of uncertainty (σ) regarding the economic viability of the attacker exploiting the next weakest link, the defender may carry out the following actions.

- Ignore the attack and absorb the potential losses. The model does not consider public costs, so such a private loss can be seen as rational when the loss is less than the security investment demands.
- Disinvest from the enterprise, which occurs when the defender's position becomes non-viable due to costs and uncertainty.
- Specify a new defensive configuration ($\mathbf{d}_t$) as each part of the true ordering is revealed by the weakest link.

The process then repeats at each step with the attacker decision. A secure state is reached when the remaining vectors below the reservation cost of attack are protected.

In addition, the IWL includes an aspect not yet considered by our model, but worthy of note: the sunk cost of defence (λ) that is incurred when the defender chooses to change the defensive configuration $\mathbf{d}_t$ in a given round. While omitted for simplicity of our exposition, our representation of SSE process investment is not dissimilar to this aspect of the IWL at the point of deployment, and its inclusion post-deployment in the proposed model is left for future work.

Our model is concerned with the genesis of the vulnerability set addressed by models such as IWL, as well as the defender's uncertainty regarding this set. This is accomplished through the application of SSE processes, reducing the defender's σ while raising the initial security condition of the software itself (represented by the attack gradient, Δx). We consider the cost and value of flaw and bug identification prior to system integration and deployment, i.e. within the Acquisition / Development phase of the SDLC. In order to do so, we complement the IWL with a model of SSE as a defender action that removes potential vulnerabilities at a greatly discounted rate (in comparison to the costs involved at later stages). Drawing from empirical software engineering, we ask how such costs can be minimised over the entirety of the system's lifecycle.

4.2. Modelling Secure Software Engineering

Consider a software project managed by a system developer, whose goal is the most efficient investment of a security budget. Each unit of potential investment can be seen as providing two benefits. The first is a reduction in economically viable vulnerabilities — vulnerabilities for which the attacker sees a return on their investment of resources. This is accomplished either through remediation (e.g. deployment of a defence) or removal (e.g. a correction in implementation) of expected vulnerabilities, and is generally driven by a risk-based attacker model (such as in SDDL-Touchpoints [5]). The second is a reduction in uncertainty regarding the identification and ordering of vulnerabilities, under the assumption that security expertise is more adept at the former than at the latter [6]. Under the IWL model the

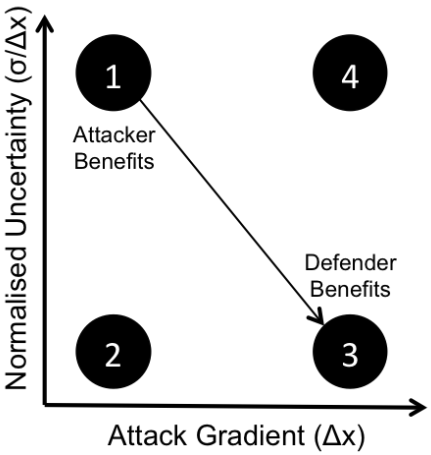


Figure 4: Investment in SSE is defined by the number of economically viable vulnerabilities (vulnerabilities for which an adversary attack results in a return after their costs), as defined by gradient of increasing attack costs (Δx) and the uncertainty regarding the nature of the vulnerabilities present (σ). The goal of secure software development investment is in the movement toward the lower right of the graph; that is, fewer economically viable vulnerabilities with less uncertainty.

attacker is always concentrated on the economically most viable vulnerability, requiring that a determination on the extent and relative weakness of vulnerabilities is known with as little uncertainty as possible in order to properly place defensive resources. As discussed in Section 4.1 this uncertainty is tightly coupled to the cost of successive attacks, such that we must consider it relative to the attack varying gradient Δx ($\sigma/\Delta x$).

Within the trade-space depicted in Figure 4, we identify four differentiating points, specified on the graph by positions 1 to 4:

1) The situation where a number of viable vulnerabilities are present, and the system developer has a great deal of uncertainty regarding the exposure resulting from these vulnerabilities (low Δx, high σ). This point arguably represents the most likely result of a modern software development effort that lacks investment into SSE processes.
2) The situation in which the system contains a number of economically viable vulnerabilities, with the system developer well aware of their existence, relative exposure, and rank order (low Δx, low σ). This corresponds to some combination of inadequate process and a lack of effectiveness in any process undertaken; in the worst case this is the 'snake-oil salesman' of commercial software.
3) The situation where there are few economically viable vulnerabilities, but this fact is not known to the system developer with any certainty (high Δx, high σ). Such a point could be a very 'lucky' development result, or perhaps a very good process leading to 'hardened' software (i.e. a high actual initial attack cost x_1) with no means to measure the result (e.g. no process artefacts, a lack of metrics). In either case, this state is likely to result in security over-investment, as the system developer seeks assurance that the resulting system is secure.
4) The ideal situation where there are few (if any) economically viable vulnerabilities in the system, and there is absolute certainty regarding this fact (high Δx, low σ). This can be considered the goal of secure software engineering investment.

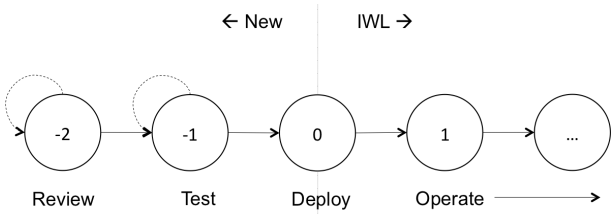


Figure 5: High level depiction of the overall integrated process, with the introduction of additional process investment steps ($t = -2, -1$) that complement the system-level security investment ($t = 1, \ \ldots, \ t_{\max}$). This is anchored by the deployment point ($t = 0$), whereby the values set by the software process are fed into the system level model (in this paper, the IWL).

This depiction is useful in conceptualising relevant security outcomes to result from investment in the software process. The secure software engineering process offers an investment opportunity for the defender on both counts, with processes such as architectural and design reviews, code reviews, static and dynamic code analysis, and risk-driven testing. As such, it is complementary to the current secure software engineering literature that places a heavy emphasis on vulnerability reduction when considering return on security investment. While this is an essential result of a secure software process, the complementary benefit — uncertainty reduction and raising the bar to attack — is just as vital to warding off security over-investment.

4.2.1. Defender Investment. Our software process model unfolds in discrete time, using negative time slices to denote pre-IWL steps for consistency of representation. A high-level depiction of the overall process is shown in Figure 5.

Investment by the defender proceeds according to two phases: Architecture and Design ($t = -2$, denoted AD), and Implementation and Test ($t = -1$, denoted IT). This investment occurs through the choice of the number of review or test iterations (i), which succeed with a probability α (finding a flaw via review) or β (finding a bug or flaw via test), respectively. Therefore, the overall investment by phase is defined as

$$I_{\{\text{AD,IT}\}} = (i \cdot c) + i \cdot (\text{eff} \cdot e)$$

where

$$\begin{aligned} i \in &\ \{i_{\text{AD}}, i_{\text{IT}}\} \\ c \in &\ \{c_{\text{AD}}, c_{\text{IT}}\} \\ \text{eff} \in &\ \{\alpha, \beta\} \end{aligned} \tag{3}$$

Here, I represents the cost of the secure software engineering activity conducted at time t for a given set of iterations within that activity, i. The value eff is the effectiveness of the SSE activity; in this case, α for reviews and β for tests. In this context, 'effectiveness' refers generally to the benefit derived from execution of the process. This is related, but not equivalent, to the effectiveness of a particular tool or activity (e.g. static analysis software or formal specification reviews) and cost c invested. Finally, e is the cost of conducting the identified fixes; for this model, it is assumed that all identified flaws or bugs are fixed. This parameter deserves further explanation, as it varies between phases.

The field of software engineering has long been concerned with the costs of fixes in different phase; Table 1 cites the findings of a survey on the topic by NASA [11]. Such surveys, along with work by those such as Boehm [10], [34], have empirically shown that costs escalate by roughly a factor of 10 with each phase of development as the project moves from requirements, to architecture and design, to implementation, and finally to operations. We reflect this by setting the in-phase cost of a fix as 0.01 (one one-hundredth of the cost of fixing a flaw in operations), consistent with the software engineering literature and the IWL unit definition for cost of defence. We exact an order of magnitude increase in each future phase for the cost of flaws (c_f) and bugs (c_b), starting from the phase they are introduced — at $t = -2$ for flaws, and $t = -1$ for bugs. Using the rule of thumb that vulnerabilities are 50% flaws and 50% bugs, investment in previous process phases is rewarded by a reduction in the proportion of higher cost fixes to in-phase fixes within the current phase. This reduction should be related to the number of iterations undertaken in the previous round, creating an incentive for early security investment reflecting current SSE thinking towards such early investment [12], [35]. This function should therefore exhibit convex behaviour, reflecting a decreasing return on security investment that is asymptotic at zero — under the assumption that both categories of error will never be fully removed. Our resulting definition for e in each phase is then:

$$\begin{aligned} t = -2: &\quad e_{\text{fAD}} &&= 0.01 \\ t = -1: &\quad \tfrac{e_{\text{fIT}}}{2^{\alpha i_{\text{AD}}}} + e_{\text{bIT}} &&= \tfrac{0.1}{2^{\alpha i_{\text{AD}}}} + 0.01 \end{aligned} \tag{4}$$

Combining equations 3 and 4 yields the following definitions for cost at Architecture and Design ($t = -2$, AD) and Implementation and Test ($t = -1$, IT):

$$\begin{aligned} t = -2: &\quad I_{\text{AD}} = &&(i_{\text{AD}} \cdot c_{\text{AD}}) + i_{\text{AD}} \cdot (\alpha \cdot e_{\text{fAD}}) \\ t = -1: &\quad I_{\text{IT}} = &&(i_{\text{IT}} \cdot c_{\text{IT}}) + i_{\text{IT}} \cdot \left(\beta \cdot \left[\tfrac{e_{\text{fIT}}}{2^{\alpha i_{\text{AD}}}} + e_{\text{bIT}}\right]\right) \end{aligned} \tag{5}$$

Here, c_{AD} and c_{IT} represent the cost of conducting a review or test, respectively. The overall cost for the software process I_{P} is then simply the linear combination of the phase costs:

$$I_{\text{P}} = I_{\text{AD}} + I_{\text{IT}} \tag{6}$$

4.2.2. Defender Payoff. The benefits to the defender that spurn investment into these secure software engineering stages is twofold. The first benefit is the reduction in the overall uncertainty faced by the defender, relative to the overall amount invested in the respective phases.

We define the overall uncertainty σ as the equal combination of the uncertainty accumulated by each phase of the software process. This can be interpreted as an equal amount of uncertainty regarding both bugs and flaws, with equal weight given to each:

$$\sigma = \sigma_{\text{AD}} + \sigma_{\text{IT}} \tag{7}$$

To determine the uncertainty reduction provided by the process elements per phase, we seek a model that correlates the number of iterations with the effectiveness per iteration. This should reflect a reduction in uncertainty that is exponentially decreasing asymptotically to 0 (just as one will never reach full certainty in practice, one will never reach fully defect-free software). Additionally, neglecting to undertake any process elements within a given phase should not result in any uncertainty reduction. We have chosen to model this as follows:

$$\sigma_t = \frac{\sigma_{max}}{2} \cdot \text{eff}^{\frac{1}{i}}$$

with

$$\begin{aligned} \text{eff} \in &\ \{\alpha, \beta\} \\ i \in &\ \{i_{\text{AD}}, i_{\text{IT}}\} \\ t \in &\ \{\text{AD}, \text{IT}\} \end{aligned} \tag{8}$$

Here, σ_{max} refers to the starting level of uncertainty, while eff and i correspond to the values relative to phases $t = (-2, -1)$ for the effectiveness and iteration counts, respectively.

The other payoff is in the increase of the gradient of attack, Δx. This can be expected to increase with diminishing returns, pursuant to the prevailing wisdom of the information security economics literature [36], [37]. We therefore seek a model that exhibits sub-linear growth and concavity, and also that reflects the unit definition for defensive costs when no investment into process is made (essentially reducing to the IWL definition). We have chosen to model this as follows:

$$\Delta x = \sqrt{(1 + \alpha i_{\text{AD}} + \beta i_{\text{IT}})} \tag{9}$$

We refer to this overall, combined model as the IWL-SSE for convenience. For the sake of clarity, this refers to the combination of two complementary models rather than a strict extension of the IWL.

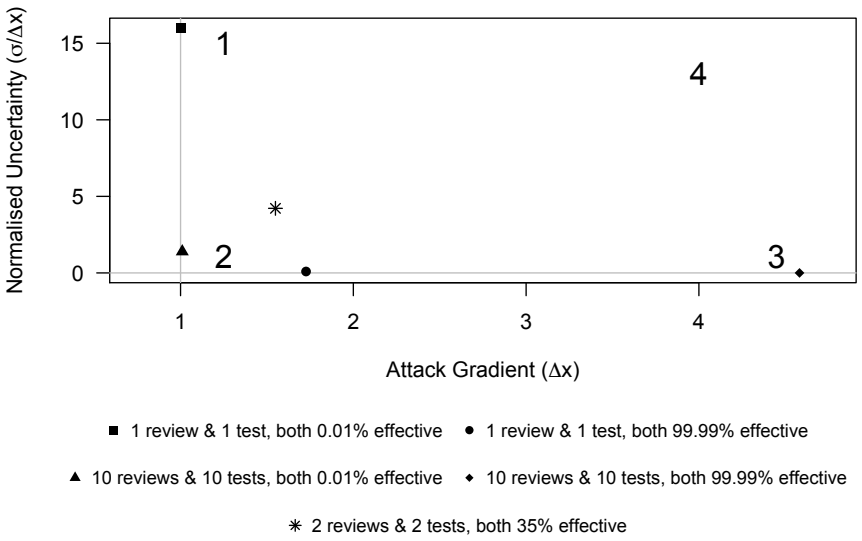


Figure 6: Example inputs applied to the IWL parameter space, demonstrating the outcome of various SSE model choices.

4.3. Model Consistency

As a first step in the evaluation of this model, we return to Figure 6 and ask if this construction faithfully represents the IWL input space. The controlling factors for the model are as follows.

1) The number of iterations the model is run at each process step (review and design), as an independent variable to be chosen by the system owner.
2) The effectiveness of each iteration, represented in the model by α and β. While also independent variables, we can look to the software engineering literature to provide reasonable estimates for these values in the absence of a particular application scenario or hard data regarding a specific SSE process. However, this is a simplification that requires further investigation, as the relationship between the effectiveness of software process activities (in their varied forms and structures) and the effectiveness of flaw and bug removal in the address of vulnerabilities is complex. In our employment of empirical software engineering, we make the simplifying assumption that the discovered flaws and bugs are intrinsically security-related, and the effectiveness of software engineering practices for their discovery are commensurate with that of 'general' bug and flaw discovery and remediation.

Recalling the discussion in Section 4.2, we seek to examine the coherence of our model with SSE investment. Figure 6 depicts combinations of input parameters to IWL-SSE under the assumption that, without any SSE process, the system developer starts with a high degree of uncertainty regarding vulnerabilities (σ) and a low attack gradient (Δx). Solid lines on the graph represent the minimum attack gradient $\Delta x = 1$ (i.e. the increase in attack costs without any process investment), and the asymptotic point of absolute certainty, $\sigma = 0$. The initial uncertainty has been set to $\sigma = 16$, consistent with the parameterisation in [6].

Comparing Figure 4 and Figure 6, we find the following relationships with IWL-SSE.

1) Point 1 is the result of a failure in the investment of process, either through a lack of iterations (low i_{AD} and/or i_{IT}), or through a lack of effectiveness in the process (α and β at or near 0%). Coupled with a high cost of review or test, this becomes the worst-case scenario of investment for no gain in protection or understanding (2 iterations, 0.01% effective).
2) Point 2 is occupied by two scenarios: few iterations with high effectiveness (2 iterations, 99.99% effective) and many iterations with low effectiveness (10 iterations, 0.001% effective). These points both demonstrate an undesirable (but not catastrophic) state, and demonstrate the concavity provided by the functional form. This is intended to reflect the decreasing returns from investment.
3) Point 3 is approached through continued investment in the software engineering process, coupled with reasonable effectiveness (driving down σ, while driving up Δx): the lower the effectiveness, the more iterations required to reach this state. Investment into more effective measures, predictably, results in the need for fewer iterations to reach the same return (10 iterations, 99.99% effective).

4) Point 4 represents an area defined by the expected minimum cost of attack, Δx. Conceptually, this space represents the lucky situation where the defender has a high attack gradient (i.e. there are few economically viable attacks), yet there is a high degree of uncertainty in this. Therefore, this value is driven by the initial Δx — as this line shifts right, the starting point for attack is increased and the points at (1) and (2) shift to the right. While we retain the unit definition employed by the IWL, future work could provide a stronger link with the role secure software process has in setting the 'hardness' of a software system against attack.
5) The final point (4 iterations each, both 35% effective) is intended to show a 'realistic' set of parameters, indicative of common development practice.

This particular form provides the general functional form desired by this model: modest return in process quickly provides returns with respect to reducing the uncertainty of the software's state, with diminishing returns — especially at low rates of effectiveness. It also reflects the need for significant, repeated investment in order to drive up the base level of economically viable attacks. While the general form fits for the purpose of exploration in this space, we leave more expressive modelling of the relationship between uncertainty and attack costs to future work.

This analysis alone fails to consider the overall return; as with post-development security, the potential for over-investment in process security exists and must be managed. Thus, we must analyse not only the overall return but also the balance of this investment between the process and operational phases. Therefore, we now turn our attention to the role of SSE investment in conjunction with post-development security investment using the IWL in order to examine optimal security investment strategies spanning the broader SDLC.

5. Results

We now examine the benefit of secure software engineering investment on the overall SDLC, with focus on the optimal distribution of security investment into the various phases — pre- and post-deployment.

5.1. Return on Security Investment (ROSI)

The solution to the original IWL is analysed in light of a specific form of Return on Security Investment (ROSI), which is defined in terms of the Annual Loss Expectancy (ALE) [38]:

$$\text{ROSI} = \frac{\text{ALE}_0 - \text{ALE}_1 - \text{average security investment}}{\text{average security investment}} \tag{10}$$

where [39]:

$$\text{ALE} = \text{Expected rate of loss} \times \text{Value of loss} \tag{11}$$

Here, ALE_0 is the ALE without security investment, while ALE_1 represents the (expectedly lower) ALE with security investment. This construction permits those making security investments a straightforward comparison between potential solutions in order to find the optimal investment strategy for a given set of initial conditions. While disagreement exists regarding the validity of annualised security benefit metrics, ROSI (and associated metrics) has emerged as a means for evaluating and justifying security expenses within an organisation [38]. Such metrics have the benefit of wide acceptance among managers and accountants, permitting security engineers and project owners to present a case for security expenditure in a non-technical, business-accessible fashion.

Applying ROSI to the IWL-SSE provides insight into the benefits that secure software process provides over the course of the SDLC, from development through deployment to operations. Figure 7 provides an ROSI comparison for a variety of parameters using the same overall scenario as that presented in [6] ($a = 1000,\ r = 5\%,\ z = 2.5\%,\ x_1 = 15,\ n = t_{\max} = 25$). For the SSE scenarios a starting $\sigma = 16$ was employed (the maximum used in IWL), along with a starting $\Delta x = 1$ (which is fixed in IWL). We employ effectiveness values of $\alpha = 60\%$ and $\beta = 30\%$ to correspond to reported effectiveness of architectural reviews [34] and singular static analysis tools [40]. However, as noted in Section 4.3, the relationship between the effectiveness of the SSE activity represented by the model and the measures of effectiveness reported in the literature for specific practices is complex; these values merely provide a reasonable starting point for analysis of the model's operation.

This figure illustrates the dual benefits of SSE within security investment situations when compared to the IWL alone. For this particular hypothetical scenario:

- *The overall return has increased in all instances employing secure software process*. This indicates that the investment in reducing vulnerabilities through investment in the software process has resulted in a more favourable position for the defender, who is able to retain a greater portion of the value of the asset (a).
- *The optimal number of proactive post-deployment defences has decreased in all instances employing secure software process*. This would imply that investment into SSE process reduces the burden on proactive post-deployment defence. Such a result undoubtedly contributes to the increased return demonstrated, as the one-time costs borne prior to deployment do not impart recurring costs.

These effects are attributable to the removal of 'weaker links' prior to deployment, which has increased the value of Δx. An important outcome is in the reduction, rather than the removal, of proactive post-deployment defences (from $k = 11$ in the optimal case of IWL under complete certainty, to $k = 3$ in the case of IWL-SSE with 8 reviews and 24 tests). Additionally, Figure 7 also highlights bounds on the return that SSE can provide, with the optimal point

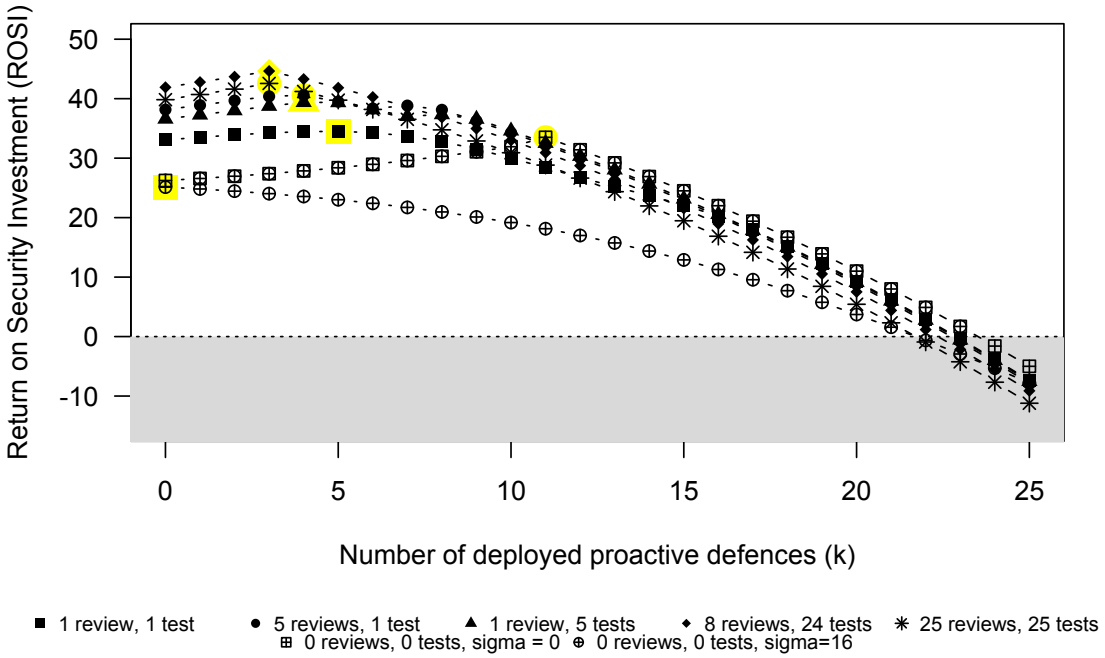


Figure 7: The Return on Security Investment (ROSI) for a variety of dynamic IWL-SSE scenarios.

(for this given set of parameters) falling below the maximum number of reviews and tests. This would indicate a 'diminishing returns' scenario that requires well-constructed process investments, rather than perpetual and unbounded expenditure. Both of these findings coincide with accepted security thinking regarding the role of SSE, and further reinforces the lack of a 'silver bullet' in information security necessitating broad and balanced approaches [12].

5.2. Return on Secure Software Process (ROSSP)

The acceptance that ROSI has experienced has led to adaptations intended to describe a number of specific security investments, notably the Return on Penetration Testing (ROPT) metric employed in [33]. We follow this convention by defining our own metric: the *Return on Secure Software Process (ROSSP)*. Formally, we define this as:

$$\text{ROSSP} = \text{ROSI}_{\text{SSE}} - \text{ROSI}_{\text{NoSSE}} \tag{12}$$

Here, ROSI_{SSE} represents the return realised after SSE investment, while $\text{ROSI}_{\text{NoSSE}}$ is the return without SSE investment. These calculations are performed according to the ROSI equation of Section 5.1. Although defined in a binary fashion by convention, this calculation is equally valid as a comparison of security investment levels (e.g., when varying the number of review or analysis iterations). This can be thought of as the ROSI of a potential state that includes secure software process against an alternative state, with the case of 'No SSE' (zero iterations of each phase) serving as a special case.

Figure 8 presents a ROSSP comparison between the previous best-case scenario ($\sigma = 0$) and SSE process scenarios, with all other values held the same as per Figure 7. Under this hypothetical scenario, at the optimal point of investment the ROSSP calculation between the IWL (with absolute certainty) and the IWL-SSE model is:

$$\text{ROSSP} = 44.6 - 33.5 = 11.1$$

Graphical results for ROSI and ROSSP across a variety of parameters are depicted in Figure 7 and Figure 8, with numeric results presented in Table 2.

Proper use of such calculations can aid the project owner in evaluating alternative development approaches, and in justifying the process investment required for secure software engineering. It is clear from this example that, while the ROSSP indicates a significant return (for this specific set of parameters), it does not constitute a 'free pass' for 'software security at all costs'. The diminishing ROSSP of the 25 review and 25 test values (asterisk line in Figure 8) show that, like any other security investment, software security must be weighed against resource commitment. While specific maxima will vary based on the costs and effectiveness of particular SSE approaches, undoubtedly the benefits wane as it becomes harder — and therefore most costly — to

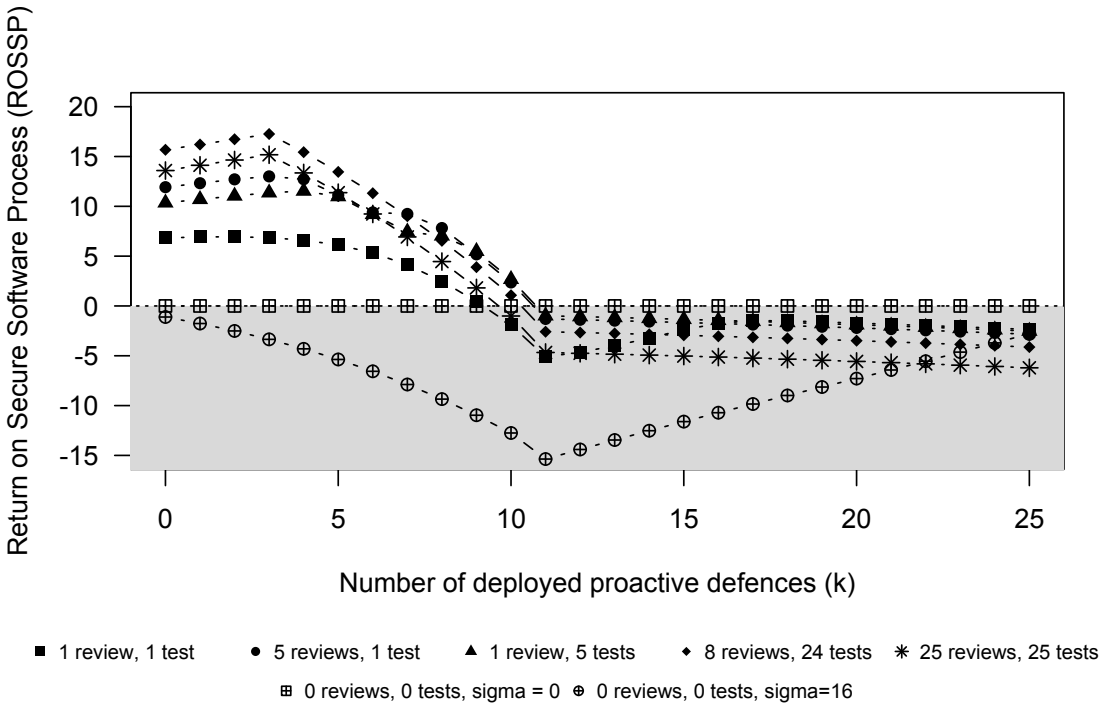


Figure 8: The Return on Secure Software Process (ROSSP) for a variety of dynamic IWL-SSE scenarios. The unfilled boxes represent the case of no process: with high uncertainty (crossed circles), and with absolute certainty (crossed squares). The latter serves as the 'best case' scenario without SSE investment, and serves as the baseline for comparison.

address the next weakest link through process alone. Where BSIMM and others have sought to identify *what* must occur to improve software security, our contribution is the first to attempt to address *how much* is reasonable. The answer to both these questions is crucial if such practices are to be adopted and rise to the importance their advocates feel is necessary to exact real change in security.

This model, and the associated ROSSP measure, is inherently conservative in that it assumes a single system deployment. In situations where the same software is to be deployed across multiple instances, the SSE investment would exhibit enormous economies of scale. A further examination of such a construction is left to future work, as it requires the consideration of parameter constellations that result in defence investment shifting toward post-deployment defence, where costs are borne at each deployment. A refined version of ROSSP and the associated analysis would consider expectations over the number of deployments.

Ultimately, the success of such metrics and models are as guiding forces, informing software development programme stakeholders and enabling rational decision-making based upon sound reasoning and explicit assumptions. It is our hope that this work opens up new lines of investigation that explicitly consider the software development process as part of the security investment, leading to richer and more descriptive models that lead practitioners toward more efficient and robust security approaches.

Reviews	**Tests**	**ROSI**	**ROSSP (IWL, $\sigma = 0$)**	**k**
None, $\sigma = 0$		33.5	–	11
None, $\sigma = 16$		25.1	-8.4	0
1	1	34.5	1.0	5
5	1	40.6	7.1	4
1	5	39.4	5.9	4
8	24	44.6	11.1	3
25	25	42.6	9.1	3

TABLE 2: ROSI and ROSSP for various configurations of secure software engineering ($\alpha = 60\%$, $\beta = 30\%$, $c_{\text{AD}} = 3$, $c_{\text{IT}} = 1$)

6. Conclusion

We have presented a model to address investment into secure software engineering processes within a software development project. Designed to represent system development lifecycle security considerations, our model is consistent with contemporary SSE practices such as architecture and design reviews, code analysis (static or dynamic), and security testing. While we have simplified these activities into two general process steps, extension to additional,

detailed process steps is a straightforward extension to the construction presented.

There are a number of avenues to further develop and expand the concept, bridging the fields of software engineering, software security and security economics. More expressive software engineering models would allow the concept to be applied to a wider range of software processes, to include modern lightweight processes. In addition, the issue raised in Section 4.3 regarding SSE activity effectiveness deserves further study, and is tightly coupled to the assumption of fix correctness cited in Section 3.2. While we account for post-introduction phase costs, research such as that detailed in [34] points to richer compositions of error type, discovery stage, vulnerability source, cost variability and effectiveness both pre- and post-deployment.

Extensions could consider additional sources of vulnerability, such those resulting from misconfigurations. Vulnerabilities of this type are typically post-development issues, addressable by activities such as penetration testing ('pen-testing'). A version of IWL-SSE that investigates the employment of pen-testing [33] provides an obvious extension point, as does incorporation of data from practices such as attack modelling and risk-based analysis into the model. A related point is the assumption that bugs and flaws correlate directly to singular vulnerabilities, when in fact this relationship is likely more complex [41].

The form of the model and the underlying assumptions characterise a security-conscious, internal development environment with a focus on the direct costs to the organisation. Commercial development (i.e. development of software for sale or as a service) introduces a need to consider process investment against the benefits of time-to-market when software patching is an option. Future work could employ models such as [42] that capture quality considerations under a 'ship and fix' mentality, and examine the impact of security considerations.

As previously stated, the choice of the IWL as the enterprise model was driven by the model input form; however, other models offer constructs for which SSE process modelling would prove a viable and enriching extension. Within this context, there remain areas where a combination of software and security research could be informative; for instance, the establishment of the initial Δx could have a relationship to the rigour of the process employed. While the functional forms for σ and Δx fulfil the required form, more specific (potentially scenario-driven) forms would benefit from ongoing studies in the study of security and may provide alternative interpretations. Our aim in applying these principles to the IWL was to examine the utility of SSE process modelling in security economic modelling; to that end, there are a number of models describing alternative attacker–defender relationships, which could also benefit from similar consideration of SSE process.

Finally, these complementary models still cover only a portion of the overall system lifecycle; a full treatment of system security investment must consider issues across the SDLC phases, and include general propositions results alongside empirical application. While more expressive and encompassing than enterprise models alone, this contribution is best considered in light of the broad and complex discussion regarding security policy, compliance, maintenance, and the conditions for system disposal.

Acknowledgements

We thank the Security and Privacy Lab at the Universität Innsbruck, and in particular Pascal Schöttle, for the fruitful discussions that shaped this research. The second author's work on this project was supported by Archimedes Privatstiftung, Innsbruck. We also thank to the anonymous reviewers for their constructive feedback.

References

[1] T. Armerding, "Is security really stuck in the dark ages?" May 2015, [Online; posted 22-May-2015]. [Online]. Available: http://www.csoonline.com/article/2925351/data-protection/is-security-really-stuck-in-the-dark-ages.html#social

[2] D. Hein and H. Saiedian, "Secure software engineering: Learning from the past to address future challenges," *Information Security Journal: A Global Perspective*, vol. 18, no. 1, pp. 8–25, January 2009.

[3] A. Austin and L. Williams, "One technique is not enough: A comparison of vulnerability discovery techniques," in *2011 International Symposium on Empirical Software Engineering and Measurement (ESEM)*, September 2011, pp. 97–106.

[4] D. Ahmad, "The contemporary software security landscape," *IEEE Security & Privacy*, vol. 5, no. 3, pp. 75–77, 2007.

[5] G. McGraw, *Software Security: Building Security In*, 1st ed. Addison-Wesley Professional, 2006.

[6] R. Böhme and T. Moore, "The iterated weakest link: A model of adaptive security investment," in *Proceedings of the 8th Annual Workshop on the Economics of Information Security (WEIS 2009)*, 2009.

[7] B. Schneier, *Secrets & Lies: Digital Security in a Networked World*, 1st ed. New York, NY, USA: John Wiley & Sons, Inc., 2000.

[8] N. C. S. Division, "Information security: System development life cycle," PDF, August 2004, http://csrc.nist.gov/groups/SMA/sdlc/documents/SDLC_brochure_Aug04.pdf.

[9] R. Pressman, *Software Engineering: A Practitioner's Approach*, 8th ed. New York, NY, USA: McGraw-Hill, Inc., 2014.

[10] B. W. Boehm, *Software Engineering Economics*, ser. Prentice-Hall Advances in Computing Science and Technology Series. Englewood Cliffs, N.J: Prentice Hall, 1981.

[11] J. M. Stecklein, J. Dabney, B. Dick, B. Haskins, R. Lovell, and G. Moroney, "Error cost escalation through the project life cycle," in *14th Annual International Symposium of INCOSE*, June 2004.

[12] G. McGraw, "Software security," *IEEE Security & Privacy*, vol. 2, no. 2, pp. 80–83, March 2004.

[13] D. Verdon and G. McGraw, "Risk analysis in software design," *IEEE Security Privacy*, vol. 2, no. 4, pp. 79–84, July 2004.

[14] G. McGraw, "BSIMM: A decade of software security," 12 2014, keynote talk to OWASP Application Security USA Conference (AppSecUSA 2014); Accessed: 2015 11 17. [Online]. Available: https://www.youtube.com/watch?v=GnlFrXPb4Qw&list=PLpr-xdpM8wG8jz9QpzQeLeB0914Ysq-Cl

[15] I. Arce, K. Clark-Fisher, N. Daswani, J. DelGrosso, D. Dhillon, C. Kern, T. Kohno, C. Landwehr, G. McGraw, B. Schoenfield, M. Seltzer, D. Spinellis, I. Tarandach, and J. West, "Avoiding the top 10 software security design flaws," PDF, 2014, http://cybersecurity.ieee.org/images/files/images/pdf/CybersecurityInitiative-online.pdf.

[16] "OWASP top 10 2013: The ten most critical web application security risks," PDF, 2013, https://www.owasp.org/index.php/Top_10_2013.

[17] G. McGraw, S. Migues, and J. West, "Building security in maturity model (BSIMM)," PDF, 2015, https://www.bsimm.com/.

[18] P. Chandra, "Software assurance maturity model: A guide to building security into software development," PDF, 2015, http://www.opensamm.org/downloads/SAMM-1.0.pdf.

[19] C. Woody and C. J. Alberts, "Considering operational security risk during system development." *IEEE Security & Privacy*, vol. 5, no. 1, pp. 30–35, 2007.

[20] R. Rue, S. L. Pleeger, and D. Ortiz, "A framework for classifying and comparing models of cyber security investment to support policy and decision-making," in *Proceedings of the 6th Annual Workshop on the Economics of Information Security (WEIS 2007)*, 2007.

[21] L. A. Gordon and M. P. Loeb, "The economics of information security investment," *ACM Transactions of Information Systems Security*, vol. 5, no. 4, pp. 438–457, November 2002.

[22] Y. Beresnevichiene, D. Pym, and S. Shiu, "Decision support for systems security investment," in *IEEE/IFIP Network Operations and Management Symposium (NOMS) Workshops*, Apr 2010, pp. 118–125.

[23] C. Ioannidis, D. Pym, and J. Williams, "Investments and trade-offs in the economics of information security," in *Financial Cryptography and Data Security*, ser. Lecture Notes in Computer Science (LNCS). Springer Berlin Heidelberg, 2009, vol. 5628, pp. 148–166.

[24] H. Cavusoglu, B. Mishra, and S. Raghunathan, "A model for evaluating IT security investments," *Communications of the ACM*, vol. 47, no. 7, pp. 87–92, July 2004.

[25] M. Cremonini and P. Martini, "Evaluating information security investments from attackers perspective: the return-on-attack (ROA)," in *Proceedings of the 4th Annual Workshop on the Economics of Information Security (WEIS 2005)*, June 2005.

[26] M. Cremonini and D. Nizovtsev, "Understanding and influencing attackers' decisions: Implications for security investment strategies," in *Proceedings of the 5th Annual Workshop on the Economics of Information Security (WEIS 2006)*, June 2006.

[27] E. Panaousis, A. Fielder, P. Malacaria, C. Hankin, and F. Smeraldi, "Cybersecurity games and investments: A decision support approach," in *Decision and Game Theory for Security*, ser. Lecture Notes in Computer Science (LNCS). Springer International Publishing, 2014, vol. 8840, pp. 266–286.

[28] S. A. Butler, "Security attribute evaluation method: a cost-benefit approach," in *International Conference on Software Engineering ICSE 2002*, W. Tracz, M. Young, and J. Magee, Eds. ACM, 2002, pp. 232–240.

[29] I. Fléchais, "Designing secure and usable systems," Ph.D. dissertation, University of London, February 2005.

[30] C. J. Neill and P. A. Laplante, "Requirements engineering: The state of the practice," *IEEE Software*, vol. 20, no. 6, pp. 40–45, Nov 2003.

[31] ——, "Requirements engineering: The state of the practice revisited," PDF, 2008. [Online]. Available: https://www.projectsmart.co.uk/white-papers/requirements-engineering-the-state-of-the-practice-revisited.pdf

[32] J. Bird, "Bugs and numbers: How many bugs do you have in your code?" August 2011, [Online; posted 24-August-2011]. [Online]. Available: http://swreflections.blogspot.co.uk/2011/08/bugs-and-numbers-how-many-bugs-do-you.html

[33] R. Böhme and M. Félegyházi, "Optimal information security investment with penetration testing," in *Decision and Game Theory for Security*, ser. Lecture Notes in Computer Science (LNCS). Springer Berlin Heidelberg, 2010, vol. 6442, pp. 21–37.

[34] B. Boehm and V. R. Basili, "Software defect reduction top 10 list," *IEEE Computer*, vol. 34, no. 1, pp. 135–137, January 2001.

[35] M. Cantor, "Calculating and improving ROI in software and system programs," *Communications of the ACM*, vol. 54, no. 9, pp. 121–130, September 2011.

[36] S. E. Schechter, "Toward econometric models of the security risk from remote attack." *IEEE Security & Privacy*, vol. 3, no. 1, pp. 40–44, 2005.

[37] J. Peeters and P. Dyson, "Cost-effective security," *IEEE Security & Privacy Magazine*, vol. 5, no. 3, pp. 85–87, 2007.

[38] R. Böhme and T. Nowey, "Economic security metrics," in *Dependability Metrics*, ser. Lecture Notes in Computer Science (LNCS). Springer Berlin Heidelberg, 2008, vol. 4909, pp. 176–187.

[39] T. Tsiakis and G. Stephanides, "The economic approach of information security," *Computers & Security*, vol. 24, no. 2, pp. 105–108, 2005.

[40] D. Baca, B. Carlsson, and L. Lundberg, "Evaluating the cost reduction of static code analysis for software security," in *Proceedings of the Third ACM SIGPLAN Workshop on Programming Languages and Analysis for Security (PLAS '08)*. ACM, 2008, pp. 79–88.

[41] F. Massacci and V. H. Nguyen, "An empirical methodology to evaluate vulnerability discovery models," *IEEE Transactions on Software Engineering*, vol. 40, no. 12, pp. 1147–1162, December 2014.

[42] A. Arora, J. P. Caulkins, and R. Telang, "Research note — sell first, fix later: Impact of patching on software quality," *Management Science*, vol. 52, no. 3, pp. 465–471, 2006.

Appendix

Notation. The following is a list of notation employed in this paper, provided for the reader's reference.

Notation	
t	The time of the phase; the model runs as $t = (-2, -1, 0, 1, \ldots)$
AD	Shorthand for the "Architecture and Design Phase" (t = -2)
IT	Shorthand for the "Implementation and Test Phase" (t = -1)
Per-phase actions, effectiveness	
i	General form for iterations: $i = \{i_{\text{AD}}, i_{\text{IT}}\}$
i_{AD}, i_{IT}	Number of iterations in review and test
eff	General form for phase effectiveness: eff $= \{\alpha, \beta\}$
α	Effectiveness of the review and test processes
Defender costs	
c	General form for in-phase costs of review, test: $c = \{c_{\text{AD}}, c_{\text{IT}}\}$
c_{AD}, c_{IT}	Cost per test iteration to conduct a review or test
e	General form for costs to fix errors: $e = \{e_{\text{fAD}}, e_{\text{fIT}}, e_{\text{bIT}}\}$
e_{fAD}	Cost of a flaw when found at review
e_{fIT}	Cost of a flaw when found at test
e_{bIT}	Cost of a bug when found at test
I_{AD}, I_{IT}	The total cost of the review and test processes
I_{P}	The total cost of the software process: $I_{\text{P}} = I_{\text{AD}} + I_{\text{IT}}$
Defender uncertainty	
σ_{max}	The starting uncertainty
σ_{AD},σ_{IT}	Amount of uncertainty remaining after review and test
σ	General from for uncertainty (consistent with original IWL): $\sigma = \sigma_{max} - \sigma_{\text{AD}} - \sigma_{\text{IT}}$
Original IWL notation employed	
a	Asset value being guarded by the defender
z	Amount the attacker loots from a upon successful attack
n	The threats to the system
Δx	The gradient of attack
$\overline{x}$	The expected attack costs
x	The actual attack costs
k	Initial (post-deployment) defence configuration
$\mathbf{d}_t$	(Post-deployment) defensive configuration
λ	Sunk costs (not employed in IWL-SSE)

TABLE 3: IWL-SSE Notation.