Security Risk Assessment and Management as Technical Debt

Kalle Rindell  
Department of Future Technologies  
University of Turku  
kkakrind@utu.fi

Johannes Holvitie  
Hospital District of Southwest Finland &  
University of Turku  
johannes.holvitie@tyks.fi

Abstract—The endeavor to achieving software security consists of a set of risk-based security engineering processes during software development. In iterative software development, the software design typically evolves as the project matures, and the technical environment may undergo considerable changes. This increases the work load of identifying, assessing and managing the security risk by each iteration, and after every change. Besides security risk, the changes also accumulate technical debt, an allegory for postponed or sub-optimally performed work. To manage the security risk in software development efficiently, and in terms and definitions familiar to software development organizations, the concept of technical debt is extended to contain security debt. To accommodate new technical debt with potential security implications, a security debt management approach is introduced. The selected approach is an extension to portfolio-based technical debt management framework. This includes identifying security risk in technical debt, and also provides means to expose debt by security engineering techniques that would otherwise remained hidden. The proposed approach includes risk-based extensions to prioritization mechanisms in existing technical debt management systems. Identification, management and repayment techniques are presented to identify, assess, and mitigate the security debt.

Index Terms—software engineering, software security, risk management, technical debt, security debt

I. INTRODUCTION

Software development processes, especially when executed following ‘agile’ or ‘lean’ principles, strive for maximum efficiency in execution [1], [2]. Features and functionality, dubbed as ‘customer value’, are pushed for as early release as possible. The value is strictly prioritized in coordination with the stakeholders, and the activities considered to produce less value pushed further back in the task execution queue, or product backlog. In practice, the quality or thoroughness of the technical solutions may get compromised. These deficiencies require the product factoring at a later time. Pushing portions of the work into a later time creates an analogy for financial debt, commonly known as technical debt [3]. While with many beneficial short-term outcomes, technical debt needs to be managed to ensure sustained development. But what happens, when the debt contains means to control security risks?

Financial debt behaves under strict and generally predictable rules. The amount of the loan is known, the period of maturity is typically fixed, and the fluctuation of interest may also be restricted. The loan givers may set a credit ceiling, restricting the total amount of debt capital for the borrower; all mechanisms that inherently protect the debtor from overburdening themselves. In the case of financial debt, the lenders compete with each other, allowing the borrower a margin. Within this margin, they can choose the loans with terms most suitable for their current goals. The loan capital, together with the interest margin, they can choose the loans with terms most suitable for their current goals. The loan capital, together with the interest each loan, has a tendency to become more expensive when it accumulates, also increasing the risk of market fluctuations or other unexpected financial risks [4].

Information security and privacy are a subject of constant and more heated public discussion. Security of computer systems is becoming more and more regulated, with infringements in the EU sanctioned for up to 20 million Euros, or 4% of the offender’s annual worldwide turnover [5]; this sum does not involve indemnity for individuals. The sanctions are an epitome of an underlying requirement for due care in managing the risks related to software privacy. Information privacy cannot exist without security, and even if the heavy sanctions concern only a small portion of software systems, security concerns all of them. A general, flexible model for identifying and managing security risk is required. The management model should cover the regulatory requirements, organization’s own security objectives, and the security requirements set by various stakeholders. Ideally, security risk is not managed separately from issues regarding the technical maintainability, robustness, reliability and quality of the software systems.

Agile, i.e., iterative, incremental, and lightweight software development projects, require security management tools and methodologies to match. Given the general technical and quality concerns, the techniques used to manage technical debt provide an approach that appears directly applicable also for security management. Security risk in software items can be viewed as a special case for technical debt, although with certain specific characteristics. To be useful for management of the software security, extensions and modifications to the current technical debt management systems are required. A key extension is the addition of a risk-based evaluation dimension to the debt items and the management framework. The security evaluation process can be used to identify unintentional technical debt, otherwise left uncovered. These technical debt items introduce new and increased security risk into the systems, effectively causing security debt.

The rest of this study is structured as follows: Chapter II provides definitions and descriptions of the central concepts of software security and technical debt, and presents the related
work. Chapter III presents the management model accommodating the security issues in various forms of technical debt; the special characteristics of security debt compared to “common” technical debt, and an approach to manage security debt are presented. Possible benefits and implications of using the debt metaphor for security management are discussed in Chapter IV. Results are summarized, and future avenues for research and implementation are discussed in Chapter V.

II. CENTRAL CONCEPTS

This chapter presents the key concepts discussed in this article. The work is based on the assumption that software development work is performed under certain security requirements, and has security targets and objectives to fulfill. Scheduling and resourcing pressures often restrict implementation, including security-related features and functionality.

A. SOFTWARE SECURITY AND RISK MANAGEMENT

Software security engineering can be defined as the part of information security that takes place during software development [6]. In the field of information security, software security mostly concerns application security, but also organizational security and infrastructure security are to be concerned [7]. Security objectives for software development are derived from the context: they originate from normative, organizational and technical contexts. Security requirements, in turn, are concrete software requirements to fulfill the objectives in the software development process.

Application security standards, such as ISO/IEC 27034 and the Common Criteria [8], [9], suggest a framework where security management is guided by a security rationale. In the context of software development, security rationale is formed by the security objectives set for the software being developed. The rationale also provides justification for the security controls to be implemented, and the security assurance to be produced.

Security objectives are derived through a risk management process. The process consists of several steps to identify and assess the security risks, and to plan for their mitigation. Risk management is a bottom-up process, and produces detailed information required to select and implement the necessary security controls in the software project. ISO/IEC standard 27005 [10] suggests a point-based system to assess the impact and probability of each identified risk. It is often necessary for the impact value to be further refined with dimensions such as amount of affected users, amount of financial loss, and other context-dependent impact factors.

Security assurance is the proof of the existence and effectiveness of the security controls. This proof is provided to various stakeholders: users, managers, technical and security experts, and e.g. auditors verifying regulation compliance. In software development, the security assurance may be produced in various phases in the development life cycle. It may cover security requirements, design, implementation and testing; at release phase, the operational security instructions and maintenance procedures are typically necessary. Security assurance is produced by various verification and validation processes, such as testing, internal and external reviews, and simulated attacks [11].

Software security is the combination of security, risk, and assurance engineering processes. These are typically guided by a Software Security Development Life Cycle (SSDLC) model, such as Touchpoints [12], or Microsoft SDL [13]. The organization may even have implemented a security maturity model, such as SSE-CMM [14], or OWASP SAMM [15]. The maturity models provide a rigorous framework for security assurance requirements, necessary for e.g. compliance purposes. In an agile software development project, concrete implementation frameworks are required to define, implement, verify and validate technical security.

B. TECHNICAL DEBT

Technical debt was initially coined by Ward Cunningham in 1992 [3]. The original intention was to just use a familiar term to explain an obfuscated term to non-technical management representatives. Afterwards, having been adopted by the research and industrial communities, the metaphor has been further developed and more mechanisms have been described for it; this includes 10 types of technical debt, and 8 distinct mechanisms for technical debt management [16].

According to contemporary consensus [16], technical debt is an analogy to the financial concept. Technical debt is comprised from a principal, an interest, and a realization probability [17]. The principal describes the quality-wise non-optimal solution residing in software artifacts. The most important affected artifacts are program code, as they deliver value to customers. In this, it is important to note that technical debt describes functionality-independent non-optimality: when a piece of code is functionally non-optimal against the requirements, it is an implementation bug – not technical debt. The age of the system has been identified a key factor in accumulation of technical debt [18]; arguably, effective management should begin as early in the software life cycle as possible.

The interest is accumulated by the debt’s principal. Interest described for a particular technical debt item, the additional work that non-optimal artifacts cause when they are used as the base for further solutions. Examples of this include, using the interface of a module in a more complex manner that required, due to the interface’s lacking description or incorrectly chosen data types; the user needs to commit additional work in comparison to using the optimal solution.

The realization probability describes the chance that a technical debt item accumulates additional work. This metric is crucial for successful technical debt management, and needs to be evaluated dynamically at decision-time. If the development plan does not make use of technical debt carrying artifacts, then the probability is zero and technical debt does not need to be paid–either as interest as previously described, or as refactoring to remove the principal. If the probability is greater than zero, different methods can be used to evaluate if paying interest is more viable than removing the principal itself.
Interest is generally lower if calculated for single iterations. Over multiple iterations, the compound interest may grow to exceed the amount of original principal, as the size and complexity of the software product increases. Hence, efficient technical debt management includes estimations of both principle and interest. These estimates, combined with estimations of organization’s capability for repayment, are used set a limit for the total amount of debt, and to properly schedule its repayment.

There is a clear distinction between debt and features that simply remain unimplemented in the backlog [20]. Debt principal potential grows at each iteration increasing the code-base, and interest accumulates following every action related to the principle, but postponing repayment. Intentional architectural debt may be a considerable time-saver in the first releases. However, later repayment of the principal and the interest – refocusing on the issue, doing things properly, and refactoring all the related components – has the potential of consuming the capability to build further features completely. This creates an inherent risk component into technical debt.

The risk that comes with having technical debt is typically considered analogous to financial risk, concerning the payback timing and work load estimations. Risk management process has been performed based on estimates of accumulated debt and its interest, not on the direct technical risk caused by the sub-optimal solution pushed to production use. Additional security risk may be present in software that perfectly fills the functional requirements, especially when it is known to be imperfectly built. It is evident that specific security risk considerations are essential in the management of technical debt.

III. SECURITY DEBT

The approach to define security debt is two-fold: First, the term is applied to that form of technical debt that has been identified through security verification or validation methods. Second, security debt may incur through technical debt (sub-optimal internal quality) in a security critical software component.

A. Related Research

Technical Debt management as a way to address security issues is a novel research topic. Security has been identified as part of quality issues, but in studies regarding technical debt, security is not addressed. Studies that mention security, treat it as another quality objective. Some studies state the technical debt principal being a liability to business [22], a central impact of security vulnerabilities. However, security issues differ from other technical debt issues in a fundamental way: security vulnerabilities can be intentionally exploited for malicious purposes. Although caused by similar bad coding practices or structural deficiencies, it is the difference in the activation mechanism, and the severity of consequences, that promotes the importance of mitigating security risk by managing the technical debt containing it.

The subject of software security is an active subject of research. This field of research is, among other things, concerned with practical solutions used to manage security risks. The base of software security work is in security awareness. This is exemplified by a survey from 2016 [23], stressing the importance of security awareness in software design activities. Similarly, security awareness has been found to have direct positive effects in software security requirement definition and in fulfilling those requirements [24]. Software security standards are an important security driver for organizations, and prompts them to defining techniques and guidelines to elicit the security requirements [25]. This also applies to Software as a Service (SaaS) instances, deployed to public computing services [26]. Security requirements cover the whole software development life cycle [27], from design to implementation and testing. Software security development life cycle models have been a subject for research especially in the context of agile development (see e.g. [28], [29]) and continuous delivery models (see e.g. [30], [31]), providing concrete solutions for the development of more secure software.

Software security design techniques cover various UML-based and other security architecture models, guidelines, principles, and patterns [32]. Threat modeling is a central design technique in the Microsoft SDL [33], a common security development life cycle model. Threat modeling techniques have their flaws, such as high time cost due to manual processing. A serious shortcoming of threat modeling is the very high amount of false negatives: the number of overlooked security flaws of Microsoft’s STRIDE has been reported as high as 64—69% [34]. This is very likely linked to both of the observed issues – time restrictions, and manual processing requiring a high level of security expertise to be combined with familiarity with the used technologies, topped with domain-specific knowledge. Despite these shortcomings, STRIDE is still the most used technique in the industry [35]. Along with the general risk analysis process, threat modeling techniques are a primary source of the items to be monitored in a security management framework. Catching bugs and vulnerabilities early in the development cycle has the effect of lowering the cost of software development.

Verification and validation techniques cover the security design and implementation in the form of security reviews, testing, audits [11]. In addition to providing security assurance, the security testing techniques are used to verify the effectiveness of the technical security controls [36]. After the software has been accepted for use and deployed to operations, security items may still be propagated to software development backlog. Continuous delivery models can be supplemented with appropriate security monitoring and incident management processes, to achieve quicker and more efficient ways to repair security issues and recover from security incidents [31].

Many technical debt management tools promote “increased security” without specifying how this is achieved. In an extensive study from 2015, only one technical debt tool – IBM Rational – was specifically identified to handle security issues in source code [16]. A recent study regarding technical
Before the security risk can be managed as debt, it has to be quantified and defined in compatible terms. This requires identification of the debt sources, and the mechanisms how the debt is accrued; a similar understanding is required about software security engineering techniques and management processes, and the characteristics of the security flaws. The key concepts of technical debt, its management, and the related security management and security risk control elements are presented in Figure 1.

In this diagram, the sources for a security debt are categorized to internal and external. The reasons for accumulation range from lack of skills or ignorance. Unintentional debt may be caused by unawareness of security issues, or ignorance to them. Debt may also be taken on intentionally, by rationally justified business reasons or sheer recklessness. The latter may also effectively rewarded when the risk never realizes and the debt doesn’t affect the customer, or the end user. Recklessness is arguably a subjective notion of the amount of accepted business risk. In case of security risk, the interest of the debt may get unacceptably high. To make an informed decision to take on the debt, the security risk must be properly identified and assessed. As a risk management strategy, the security risk is then accepted and the debt principal increased. Another problematic accumulation mechanism is the unintentional risk, leading to unrecognized debt: the ultimate goal of software security management should be nothing short from preventing this type of security risk altogether.
Technical debt management requires a set of distinct actions to recognize debt and determine its handling, in the center of the diagram [16]. When enhanced with security risk management (on the right), this list must be amended by security training: Arguably all engineering requires specific training, and software security is a prime example of this principle. Awareness is a key enabler for the security processes. Various security engineering techniques can be used to identify the unintentional security debt, subject to the management activities. These include reviews of policies, design, code, and configuration; static and dynamic reviews; various forms of security testing and simulated attacks; or professional observations, interviews and reviews, aiming to reveal and pinpoint potential security risk [11]. These are all included in the security management activities, to be integrated with debt management actions.

Identification can take place in any stage of software life cycle: optimally, security vulnerabilities are identified while the software is still in development, before the security vulnerability has become potentially exploitable. Proper security architecture and design and thorough risk identification, assessment and mitigation processes are part of this activity. Some security flaws in code or configuration may also be found in security testing performed on top of regular quality assurance tests. The flaw is then measured, and appropriately prioritized based on the risk assessment. Prioritization is essential, as simply assessing the principal and debt of a technical debt item may not provide sufficient information about the implications of a particular sub-optimal security solution. Risk assessment also affects the selection of prevention, i.e., risk mitigation activities. These can be implementation of security controls, transferring the risk to a third party, avoiding it altogether by choosing not to implement the feature – or accepting the risk, thus increasing the intentionally accumulated security debt.

Security management activities include creation of appropriate security assurance to measure and monitor the accumulation of security risk: In this area, the technical debt management activities consistently overlap those of security management. In cases when the existence of security risk is know, the debt is documented and monitored. These both form a part of the debt’s interest. The main security debt item is communicated, often in the form of a security operations guide, or an incident plan. The main point of this activity is to make the risk visible and enable discussion about it. To complete the analogy with debt, finally the repayment occurs. This takes place in the form of refactoring the sub-optimal solution (the principal) and the dependent modules and configurations (the interest), or in the form of a write-off, if for example the solution’s life-cycle ends, nullifying the debt.

C. Security Debt Types and Repayment

Several types of technical debt have been identified: in a systematic mapping study, 10 distinct debt types were identified [16]. Typically in debt, as in software security, the code items are central, as they deliver the required functionality. However, a summary of software defect studies by NIST claims only 20% of software flaws are created at the implementation stage, whereas 70% are created in requirement and architectural design phases, and the remaining 10% in integration [38]. The same analysis shows that 50.5% of the defects are found in integration phase, and about 21% are discovered in the operations and maintenance phase. Only 9% of the defects are found in QA testing. The cost of a defect found in operations is projected to be 110 times higher to fix compared to a defect found in requirement review; in testing the cost is 16 times higher. A similar study of security-related defects remains yet to be conducted. Results may be assumed to be similar, although the cost of security flaws nearly always includes secondary effects as well, thus adding a ‘non-technical’ interest component into the technical security debt.

Security is traditionally been heavily reliant on verification and validation. Extensive security testing by several types of techniques are adequate to reveal the most typical security vulnerabilities; monitoring and incident response cover the residual security risk. Effectively the debt is identified by the same means it is mitigated, the act of repayment following
a deliberate decision to do so. An example of technical debt accumulation, measured as security risk, and its repayment is given in Figure 2.

In the diagram, technical debt is expressed as security risk, creating security debt. An amount of security risk exists all the time, and it increases as the code base grows. It also forms the lower bound for the total security debt: mitigation efforts reduce the total risk in software, but cannot go below the combined risk level of the principal and compound interest. Repayment takes place on two occasions: First repayment occurs as the debt ceiling is about to be reached. The ceiling represents the maximum amount of acceptable security debt, or risk. The repayment activities also reduce the interest, although it is shown to grow as the code base grows, gradually increasing the complexity. Reduction of total security risk requires revisions in the risk management strategy, representing organizational and technical changes often not feasible in an ongoing software development project. In software development, a portion of the security risk can be taken to represent legacy code and components.

The portfolio approach [19] provides a basic framework suitable also for managing the security risks. In the core of the model lies a security debt item: such an entity has characteristics of both security risk items, and technical debt items. The items consist of identifying information; a responsible person; description of the item; and work estimates for both the principal and the interest. In forming the debt portfolio, two important estimates are added into the items: standard deviation for the interest, and correlation with other debt items on the list. Items in the security debt portfolio need additional input from risk management: a combined value of risk probability and impact as proposed in ISO/IEC risk management standard [10], or a similar quantitative risk assessment measure.

Adding security risk dimension to the item requires both the ability to identify and assess security issues: it also integrates the security risk evaluation right into the software development backlog management process. In practice, the evaluation process depends on the type of the debt item. In Table I a compiled list of 10 technical debt types, as identified in [16], is mapped to common security engineering and security assurance techniques. These techniques are derived from software security development life cycle models [12], [13]. The risk value for a Code-type debt item can be a result of a review, issue found in static or dynamic scanning; Code-type debt items can also be found through testing, integration or operations and will be added to the list as they are identified and assessed. Each item is assigned a weight based on its interest, relation to other items, and the security risk. The total weight of the portfolio (i.e., debt list) represents the total amount of managed debt.

The weight given to a security debt item is valued at decision time. Hence, a pre-process step to the suggested portfolio management approach is required. Prior to the decisions, the target weights are identified, and their interests, relations, and risks themselves are evaluated to ensure the weights are representative of the project’s current state. Pre-processing is a laborious step, but it can be considered less taxing than a mid-development security risk assessment as decision targets limit the set of weights that are to be reassessed.

In a software development organization utilizing contemporary work load estimation and monitoring techniques can plan the mitigation for each development iteration by scheduling certain amount of refactoring into the iteration backlog. A prioritization model explicitly taking security risk into account makes the security issues visible and explicit. This approach should prevent improper prioritization the security-related issues: omitting them accrues a risk invisible to standard technical debt management, and over-prioritization is mismanagement of the software development resources.

In terms of technical debt, a security incident would represent a forced payment of the loan. In certain cases, the security flaws can be covered by external means, e.g., by improving network security. This mitigation strategy – reliance on infrastructure security, external to the software, instead of building security into the software itself – forms a part of the interest, and increases the total cost of ownership significantly. To avoid unintentional debt, and to effectively manage the intentional security risk, a risk management framework is required. The financial risk management framework – the one that technical debt models typically refer to – needs this information: security debt may fall due unexpectedly and unplanned, in the form of a security incident.

<table>
<thead>
<tr>
<th>DEBT TYPE</th>
<th>IDENTIFICATION METHODS</th>
</tr>
</thead>
<tbody>
<tr>
<td>Requirements</td>
<td>Requirement reviews, threat assessments</td>
</tr>
<tr>
<td>Architectural</td>
<td>Architecture reviews, security and quality standards</td>
</tr>
<tr>
<td>Design</td>
<td>Design reviews, tools, security and quality standards</td>
</tr>
<tr>
<td>Code</td>
<td>Training, code reviews, static and dynamic analyses, security testing</td>
</tr>
<tr>
<td>Test</td>
<td>Test case reviews, independent validation, witnessed tests</td>
</tr>
<tr>
<td>Build</td>
<td>Configuration management reviews, tools</td>
</tr>
<tr>
<td>Documentation</td>
<td>Documentation reviews, audits</td>
</tr>
<tr>
<td>Infrastructure</td>
<td>Audits, reviews, tools</td>
</tr>
<tr>
<td>Versioning</td>
<td>Tools, reviews</td>
</tr>
<tr>
<td>Defect</td>
<td>Incident monitoring, tools</td>
</tr>
</tbody>
</table>

IV. DISCUSSION AND FUTURE WORK

The metaphor of debt describes many aspects of software development with considerable accuracy. The same applies to
software security, and especially the management techniques. However, there are certain key differences:

1) Prioritization of debt. Much effort has been put into estimating the principal and interest of technical debt, but also to setting the priorities for repayment schedule. Security items are assessed primarily by risk, with inherent uncertainty especially in the probability of the risk – impact is usually better known. The security incidents also have a multiplicative effect, further complicating estimations. III-prioritized security items lead to inefficiencies, or, in worst case, security incidents and forced payment of the debt, with potentially heavy interest. Comprehensiveness and correctness of risk identification, and the accuracy of risk assessment are crucial for the prioritization of debt items.

2) Indirect consequences. Security controls in software are typically used to protect the software itself, as well as other assets. Loss of data confidentiality or integrity, or even the availability of the service the software produces, may lead to severe secondary effects for the stakeholders. The effects of a security incident are typically system-wide, resulting to process disruptions and data loss.

3) Compound effect. A single security flaw may remain hidden for indefinite period of time, and may not alone be enough for an attacking party to achieve a security exploit. Combined vulnerabilities, however, are more likely to result in loss of confidentiality and integrity (see e.g. [39]).

4) Interest. The interest of a money loan is generally defined by written contract and only partially subject to market changes. Technical debt interest is based on work estimates, and includes much more inherent uncertainty in interest calculation. Increased uncertainty by adding a new security risk estimation into the interest calculation may lead to improper prioritization of security-critical items. Security risk must be estimated accurately and in terms equal to other variables used for total debt calculation.

In the financial Modern Portfolio Theory (see [40]), the value of the portfolio is based on the weighted value of each item in the portfolio. This poses the greatest challenge to the management model: money, stock and even lines of code can be measured, but all software development work contains an inherent amount of uncertainty. The accuracy of estimation techniques may be highly variable [41], risk estimation further increasing the uncertainty. A truly refined model needs to be tuned and micro-managed within the capabilities and constraints of each particular software development organization. In appreciation of the complexity of this work, the framework presented in this paper merely contains the building elements, and suggestions for their use. Although methods to assess the total technical debt principal in the codebase have been introduced [22], the accuracy of the debt estimation for due repayment remains a central theme to be addressed. This includes the related issue of repayment prioritization.

Future work regarding security properties of technical debt should address identification and repayment of the debt types essential for security work in software development. Main focus is be directed to security architecture and design. These activities belong to the field of software security engineering, which covers the life cycle stages prior to and including deployment of software releases. Application and infrastructure security experts deal with the security incident response and other external security measures taken in operations and maintenance. This may contain items such as architectural knowledge management – including potential security issues and vulnerabilities into the architecture and design itself, effectively bringing threat modeling an integral part of software design work. Integration with coding and testing, and automating the processes, are essential for efficient and effective software security management.

V. Conclusions

The management of security issues in software development using technical debt management methods provides interesting approaches for both research and practice. Suggested portfolio-based approach extends the technical debt management by adding security risk evaluation process for the technical debt items. The proposed approach fundamentally changes the debt calculation and repayment prioritization mechanisms. It also has the potential of increasing awareness of potential security issues in the software being developed.

A central benefit of security debt concept is its familiarity to the software developers. The concept of technical debt is well known, and technical debt management models are widely used in software development organizations. However, the existing technical debt models address the underlying security risk inadequately, or not at all. This makes extending the management models and tools with security features, and involving a security risk management process to debt items is essential. Including the security effect into debt calculation strengthens the management of security risk in software development processes, and helps creating more secure software. Security engineering and assurance techniques may also reveal technical debt previously left undiscovered, by finding items that meet functional specifications, but contain security vulnerabilities that need addressing.

Managing the security risk as debt provides new means for security risk mitigation in software development. Security debt can be directly integrated into existing technical debt management frameworks and tools with few technical adjustments. The more significant modification is the addition of a security risk management process. Bringing security management closer to the management of software development provides the developers and security experts a direct tool for communication. The promise of the proposed approach is increased efficiency, better predictability over future workloads, and more secure software done with better quality.