Security and Software Engineering: Analyzing Effort and Cost

Callum Brill, Aspen Olmsted
Department of Computer Science
College of Charleston, Charleston, SC 29401
Email: brillch@g.cofc.edu, olmsteda@cofc.edu

Abstract— There are many systems developed to model and estimate the software development lifecycle of a product, such as Constructive Cost Model (CoCoMo) II and SEER for Software (SEER-SEM). As the demand for security in software engineering rises, engineers are proposing changes to the development lifecycle to better integrate security. These changes in the Software Development Lifecycle (SDLC) come with the need for changes in how we model the associated costs. Specifically, this paper analyzes the costs of a Web Content Management System with regards to security and proposes adjustments, based on lifecycle changes, to the CoCoMo II cost model that would allow for security to be better factored into project management.

Keywords- Software Engineering; Cyber Security.

I. INTRODUCTION

The cost of software development projects can be quite difficult. The Software Development Lifecycle (SDLC), the lifecycle software engineering project undergoes, consists of the following stages [1]:

- Analysis–Developing the goals to be achieved by the software and defining the scope of the software with regards to those problems to ensure that the project does not fall victim to scope creep.
- Requirements–Translating project goals into concrete operations of the software.
- Design–Formulating detailed descriptions of the previously defined operations, including but not limited to the design of user interfaces, internal logic decisions and modeling system interactions.
- Implementation–Encoding the agreed upon design choices into a working software application.
- Testing–Evaluating the correctness of the implementation to remove potential defects.
- Deployment–Deploying the tested implementation into a production environment so that the software may be consumed by end users.
- Maintenance–Resolving issues that arise during use by consumers, ensuring that the software can continue to be used and keeping the software from becoming obsolete. This is typically the longest stage and is an ongoing effort.

However, this lifecycle is becoming less appropriate for representing software development, as security is becoming more important. Many of the existing models to estimate cost are based on this lifecycle, which means the need to update those cost models rises along with the need to replace the SDLC with a more secure process.

Another cost that models do not account for is Information Technology (IT) and technical debt. IT debt is the idea that systems can accrue liability over time, usually by having maintenance operations postponed or added to an ever-growing backlog; and if that liability is not recognized and dealt with, it can grow exponentially [2]. Similarly, technical debt is the liability that one assumes when producing software products and deciding to produce code that may not necessarily be the most optimal solution in the hopes that it will ease schedule pressure [3]. In both cases, this liability may be reduced by devoting man-hours to either, in the case of IT debt, performing maintenance tasks from a back log or, in the case of technical debt, refactoring, i.e. changing code without changing the external functionality. With the potential to become wildly expensive, it is important to incorporate the potential of these debts into cost models.

Our paper examines the position of IT and technical debt in the current software development lifecycle and cost models, as well as changes to the SDLC, and proposes factors to better estimate the amount of effort necessary to resolve these issues.

The organization of this paper is as follows. Section 2 describes related work and the limitations of current methods. In Section 3, we give a motivating example from which we draw our information. Section 4 describes our proposed changes to current models and methodologies. Section 5 contains the conclusion and possible future work using our models and the field of secure software engineering.

II. RELATED WORK

A. Constructive Cost Model

There are multiple models used to estimate the cost of developing software: The Constructive Cost Model (CoCoMo) [4] and its offshoots, such as SEER for Software [5] (SEER-SEM) and the numerous in-house models used by software development firms. CoCoMo II, the model proposed by Boehm et al. [4], is designed to consider a shift in development paradigms away from waterfall development and towards iterative patterns, such as agile and extreme programming. CoCoMo II has various factors that determine cost, including a reliability factor, however it has no factor indicative of security development costs. Prior versions of CoCoMo had a factor related to security; however, it was an effort modifier that dealt with the development of classified software. The shift to cover software built on off the shelf platforms [4] has resulted in the removal of such security factors. The driving motivation in the shift is the belief that the platform will be secure; therefore, any software built upon it will be secure. Our paper suggests factors to estimate the cost of developing software using a secure process. Madachy has developed a Web application to using CoCoMo II to be used to estimate costs [6].

B. Effort Cost And Reduction

Using the platform as a service (PaaS) model is a common method of saving on costs as it removes the need for end-users to develop from scratch. Olmsted et al. estimate the total cost of an platform to be approximately 13 million dollars by using a metric consisting of a measure of source lines of code (SLOC) and a trace of code execution *[7]*. We use methodologies from this analysis to estimate cost factors related to the security of these platforms and the hypothetical cost to have been developed using an alternate lifecycle.

C. Secure Sofware Development Lifecycle

There are many proposed enhancements for the Software Development Lifecycle from many different sources. Microsoft advocates a secure development lifecycle to complement the security of their operating system. Microsoft's proposed Development lifecycle add several stages to the development lifecycle, including [8]:

- Security Education and Awareness – Ensuring that developers are educated on the ideals surrounding security.
- Determining Project Security Needs – Analyzing if the project has a crucial need to follow the Secure Development Lifecycle
- Designing Best Practices – Fitting common best practices to your project and determining new best practices as necessary.
- Product Risk Assessment – Estimating the appropriate amount of effort to create an appropriate level of security.
- Risk Analysis – Analyzing possible threat vectors.
- Security and Best Practice Documentation and Tooling – Creating tools and best practices which can be easily followed by an end user to help ensure the security of their environment.
- Secure Coding Policies – Following prescribed methodologies in order to prevent poor implementations of design e.g. avoiding certain functions, leveraging compiler features, and using the latest version of tools.
- Secure Testing Policies – Applying secure testing policies in order to verify the security of you application. This does not make the product secure, only verifies that it is.
- Security Push – Pushing to ensure that any legacy code that is used is secure.
- Security Audit – Determining if the product is appropriately secure to ship to consumers.
- Security Response and Execution – The creation, and execution if necessary, of plans with which to respond to security breaches.

Some of these steps are relegated to technical debt and often not handled at the appropriate points in development and cut due to cost, especially in agile development and commercial off the shelf products. With these steps in mind, our paper our paper provides a factor to add to CoCoMo to estimate the cost of integrating these procedures into a development lifecycle.

III. Motivating Example

WordPress [9], Drupal [10], and Joomla! [11] are three of the most widely used COTS Web platforms. These platforms allow end-users to create Websites with significantly less effort than creating their own Website; however, Websites running these platforms are among some of the most exploited on the internet due to the low barrier of entry. Our paper examines one of these platforms, Drupal, in order to determine factors that should be added to CoCoMo II in order to adequately cover the costs of secure development.

According to work by Meike, Sametinger and Wiesaur, Joomla and Drupal both have serious design flaws that place the platforms at definite risk. Currently identified flaws include the allowance of file uploads with unchecked contents, the existence of HTTP headers that contained data capable of being manipulated and escalations of privilege. These flaws in the code exist because of flaws in the design process [12].

IV. Contribution

We determined the factors to add to CoCoMo II through an analysis of an unsecure, obsolete version of Drupal. This analysis is an examination of the number of lines of source code involved in flaws in the platform. We determine that that is the cost of the flaw, valued in source lines of code (SLoC). We then run a similar analysis on the latest version of Drupal. We then compare those costs with the cost of the newest version and compare vulnerabilities to determine if the design flaws still exist. Here all costs are equivalent to the number of lines of source code, so our calculation can give us a comparable measure.

In Table 1, we have an explanation of several pain points and security vulnerabilities in two common Web content management systems Drupal and Joomla! in versions 5.2 and 1.0.13, respectively. In this table ✓indicates and issue that is not present in the software, ✗ indicates an issue that is present in the software and an ! indicates that the issue has been partially resolved in the software. Using these unresolved and partial resolved issues, we measured the number of lines of code per function call, using a PHP module called xDebug.

Table 2 contains a list of flaws, the status of that flaw in Drupal 5.2, the number of lines of source code necessary to achieve the functionality present in Drupal 5.2, the status of the flaw in Drupal 8.2.3, the number of lines necessary to achieve the functionality in Drupal 8.2.3, and the technical debt balance. In this table the technical debt balance is a value based on whether or not the flaw had been resolved. Should the flaw have been resolved, the SLoC from the obsolete version of Drupal is subtracted from the SLoC of the later version of Drupal. Should the flaw not have been resolved, the amount of technical debt is represented by the SLoC of the current version of Drupal.

Examining the measurements, based on a measurement of flaws present in Drupal version 5.2 and 8.2.3, in Table 2 we can see that some of the more serious flaws that were present in 5.2

TABLE I WCMS VULNERABILITIES [8]

	Joomla!	Drupal
Community		
Security patches	✓	✓
Reporting of vulnerabilities	✓	✓
Hints to countermeasures	✓	✓
Installation		
Security hints	✓	✗
Security settings	✗	✗
Parameter Manipulation		
HTTP header data	✗	✗
Super global arrays	✓	✓
Cookies	✓	✓
Remote Command Execution	✓	✓
Forms	!	✗
Cross-Site Scripting		
APIs against XSS	✓	✓
XSS via URL parameter	✓	✓
XSS via search fields	✓	✓
XSS in other forms	✓	✓
XSS in back-end	✓	✓
SQL-Injection		
Any countermeasures	✓	✓
User Administration		
Login: XSS or SQL-Injection	✓	✓
Secure Passwords	✗	!
Sessions at the Server	✗	!
Sessions at the Client	✓	✓
Session Hijacking	!	✗
Access to Functions	!	✓
Spam		
Contact forms like CAPTCHA	✓	!
Spam Relays	✓	✓
E-mail addresses	✓	✗
Malicious File Upload		
Checking File Endings	✓	✓
Checking File Contents	✗	✗
Elevation of Privilege		
Privileged users	✗	!
Administrators	✗	✗
Optional Modules		
Warnings	✓	✓
Security measures in core	✗	✗

were resolved. These flaws allow authors or users who had escalated their privileges to that of an administrator to post code directly into Webpages. In our Drupal 5.2 test environment, we executed code that showed the server information, but it would be fully possible for a malicious user to deploy a Web shell through these vulnerabilities. There is also a clear difference in the overhead code between the two versions. For example, during the installation the obsolete code required 638 lines of code, while the modern version required 4616 to execute that same function. It is clear, however, that even though some issues are resolved there are several issues that remain and would need to be resolved through the effort of the end user.

TABLE II ANALYSIS OF COST PER FLAW

Security Feature	Drupal 5.2 Status	Drupal 5.2 SLoC	Drupal 8.2.3 Status	Drupal 8.2.3 SLoC	Technical Debt Balance
Security Hints during Installation	None	638	Hints present	4616	3978
Installation Security Settings	None	638	None	4616	4616
Secure Passwords	Not Enforced	860	Not Enforced	4694	4694
File Content Scan	Does not scan file contents.	798	Incorrect file types are detected	1566	768
Administrators	Can Execute PHP at will	907	PHP not Executed	1953	1046

Using the data gathered from our Drupal test environment, we have developed two factors which increase the accuracy of the CoCoMo cost Models to reflect the true costs of developing secure software. The first factor, a multiplier of 3.47, is applied to greenfield engineering projects to estimate the effort of designing a project using a Secure Software Development Lifecycle (SSDLC) rather than using the standard SDLC. This value was calculated by comparing the SLoC of flaws which had been resolved between versions of Drupal (8135 lines / 2343 lines).

The second factor, a multiplier of 1.607, should be used to calculate the effort that will be needed to handle technical debt when a software product has already been developed and the development team did not use a SSDLC. This multiplier was determined by comparing the technical debt balances of the unresolved flaws with the technical debt balances of the resolved flaws (9310 lines / 5792 lines).

V. CONCLUSIONS AND FUTURE WORKS

In this paper, we have proposed an additional factor to CoCoMo II. This was done by calculating and comparing the cost of code in flawed portions of a Web platform and a less secure, obsolete version of the same platform. We believe that the use of these factors would accurately describe the amount of additional effort necessary to integrate a SDLC as well as the possible pitfalls that arise as technical debt.

Future works may include the analysis of several other off the shelf Web content management systems, using the same analytical method, to increase the size of the data set and consequently the accuracy of the secure development factor or the development of such a factor for other costing models.

REFERENCES

[1] I. Sommerville, Software Engineering, Harlow: Addison-Wesley, 2001.

[2] D. Britton, "Why IT Debt is Mounting," Micro Focus, 22 09 2014. [Online]. Available: http://www.networkworld.com/article/2686761/it-skills-training/why-it-debt-is-mounting.html. [Accessed 27 9 2016].

[3] P. Kruchten, R. L. Nord and O. Ipek, "Technical Debt: From Metaphor to Theory and Practice.," *IEEE Software*, vol. 29, no. 6, pp. 18-21, 2012.

[4] B. Boehm, B. Clark, E. Horowitz, C. Westland, R. Madachy and R. Selby, "Cost Models for Future Software Life Cycle Processes: COCOMO 2.0," *Annals of Software Engineering*, vol. 1, no. 1, pp. 57-94, 1995.

[5] D. Galorath, Galorath, [Online]. Available: http://galorath.com/products/software/SEER-Software-Cost-Estimation. [Accessed 15 April 2017].

[6] R. Madachy, "CoCoMo II - Constructive Cost Model," [Online]. Available: http://csse.usc.edu/tools/COCOMOII.php. [Accessed 27 9 2016].

[7] A. Olmsted and K. Fulford, "Platform As A Service Effort Reduction," in *Proceedings of The Eighth International Conference on Cloud Computing, GRIDs, and Virtualization (Cloud Computing 2017)*, Athens, 2017.

[8] M. Howard and S. Lipner, The security development lifecycle, Redmond: Microsoft Press, 2006.

[9] Wordpress.org, [Online]. Available: https://wordpress.org/. [Accessed 16 April 2017].

[10] Drupal, [Online]. Available: https://www.drupal.org/. [Accessed 2017 16 April].

[11] "Joomla!," Open Source Matters, Inc., [Online]. Available: https://www.joomla.org/. [Accessed 2017 16 April].

[12] M. Meike, J. Sametinger and A. Wiesaur, "Security in Open Source Web Content Management Systems," *IEEE Security and Privacy*, vol. 7, no. 4, pp. 44-51, 2009.

[13] A. Olmsted and K. Fulford, "Platform As A Service Effort Reduction".