Closing the Feedback Loop Between UX Design, Software Development, Security Engineering, and Operations

Jessica Nguyen
Marc Dupuis
jnguy@uw.edu
marcjd@uw.edu
University of Washington
Bothell, Washington

ABSTRACT

There have been many evolutions of the software development lifecycle (SDLC). These differing models have moved software development groups from sequential development to a more agile and iterative development model. Increasing awareness and research focused on the cyber security landscape has resulted in a large push for "shifting security left" in the SDLC. With security engineering teams engaged earlier and more often throughout the SDLC, security issues will be found and fixed earlier, which increases efficiency while lowering cost and overhead. While this has been an important cultural and infrastructural shift for many technology companies, there is still a gap in this feedback loop that needs to be bridged: the gap between user experience designers and the software, security, and IT/operations engineers. Tradeoffs have been made between security and usability—a challenge known as "usability versus security." Much of the research that propose how to change these two fields from opposing forces to being cross-functional allies offer simplified solutions but don't go into granular detail about solving the problem. This paper covers the evolution of the SDLC from the Waterfall model through the DevSecOps agile methodology and proposes a new development model: the Technology Development Lifecycle (TDLC). This TDLC model aims to keep designers, software engineers, security engineers, and IT/operations all within a tight feedback loop throughout a continuous integration/continuous development pipeline. We will discuss various workflows, use cases, and technologies that can be used later on to implement a working environment that can enforce the TDLC model.

CCS CONCEPTS

• **Security and privacy → Human and societal aspects of security and privacy**; *Systems security*; *Software and application security*; • **Software and its engineering → Software creation and management**; • **Human-centered computing**;

KEYWORDS

user experience design, software development, security engineering, operations, secure software development

ACM Reference Format:
Jessica Nguyen and Marc Dupuis. 2019. Closing the Feedback Loop Between UX Design, Software Development, Security Engineering, and Operations. In *The 20th Annual Conference on Information Technology Education (SIGITE '19), October 3–5, 2019, Tacoma, WA, USA.* ACM, New York, NY, USA, 6 pages. https://doi.org/10.1145/3349266.3351420

SIGITE '19, October 3–5, 2019, Tacoma, WA, USA

ACM ISBN 978-1-4503-6921-3/19/10…$15.00
https://doi.org/10.1145/3349266.3351420

1 INTRODUCTION

The study of human computer interaction (HCI) has long provided insight on best design practices to aid the user experience of products, applications, and other technological interfaces [20]. While this has contributed much to the evolution of usability design, a competing quality between the HCI and cyber security fields has garnered some attention.

The rise in awareness and further study in cyber security has faced many adoption challenges. The adoption challenge with the most attention has been injecting security into the software development lifecycle (SDLC) to create a "secure software development lifecycle" (SSDLC) or "secure development lifecycle" (SDL) [13]. The main focus area that spirals from the idea of the SDL is moving from DevOps to DevSecOps, where agile development is joined with security processes (e.g. security through design and security reviews), continuous integration, continuous development (CI/CD), as well as the IT/operational tasks that come with deploying a software product.

There have been countless proposals on how to best integrate software and security processes but something that is oftentimes overlooked here is the inclusion of the HCI and user experience (UX) designers into this cross-functional new SDL. Considering the fact that how UX design principles are applied to the end-product can greatly impact how effective/successful these added security controls will be with and for the user, it is paramount that we tighten this feedback loop with designers as another stakeholder in the development process. Adapting the DevSecOps model, we are proposing a new development model that includes designers into the feedback loop.

2 BACKGROUND

This section will be used to go over the current related work in this area. This includes work on the evolution of the SDLC to

include security and agile development, UX research/design, and the relationship between usability and security.

2.1 Evolution of the SDLC

There are different versions of the SDLC that have been adapted in order to define, standardize, and approach the process of developing software. SDLC models are focused on scoping out a problem domain, understanding the scope, defining a solution domain, outlining the implementation of software to solve this problem domain.

2.1.1 Introduction to the SDLC and Development Methodologies. The SDLC was introduced in the 1960s to help standardize the approach of developing large-scale information systems in a time where project management was difficult in large business conglomerates where parent companies owned multiple subsidiaries across different industries [7]. The SDLC is a process model laid out on how to approach designing software to solve a specified problem domain [14].

This basic SDLC model helped pave the way and standardize an approach to implementing software. However, the basic SDLC did not satisfy all development needs.

A notable adaptation of the SDLC is the Waterfall Model, which was introduced in 1970. This model was designed as a linear and sequential timeline of phases. Each stage of the Waterfall model must be completed before moving onto the next, and it is not designed for a past phase to be revisited.

The key advantage of utilizing this model is that it's easy to follow and understand. This works well if the software being developed has strict and straightforward requirements. However, since each phase must be completed prior to moving on to the next, this model can cause very slow development as well as inadvertently making the product inflexible (in terms of time and cost) to adaptation if any changes were to come up later on in the process. Due to these limitations, the Waterfall Model would not work everyone.

The V-Shaped Model is another adaptation of the SDLC. Like the Waterfall Model, this is also sequential but it introduces an additional two layers of parallel test design phases and testing phases. This increases the rigidity of the model while maintaining its low complexity. The introduction of the test design phases at each level also helps introduce additional coverage of risk analysis throughout the development process, which is where the original Waterfall Model falls short [14].

Also like the Waterfall model, the V-Shaped Model can be slow and time-consuming, which would not scale for more long-term projects. This process and all of its required test designs/testing would also be overkill for a project with a narrower scope. As the technology landscape continued to grow, the SDLC methodologies changed to better support faster, leaner, and more agile development practices.

2.1.2 The Agile Methodology and the SDLC. The technology industry fostered frustration with the costly and time-consuming SDLC methodologies. The large periods of time between requirements definition to delivery often led to project failures, cancellations, or costly overhauls if the requirements changed over time.

The software development community sought a more lightweight methodology to standardize an SDLC that would support an idealized faster, leaner, and more agile development process. In 2000-2001, a group of seventeen people in the industry met and came up with something called the Agile Manifesto. The Manifesto outlined the four key values of this new Agile Methodology, followed by twelve supporting principles to help software developers implement this new methodology. The four key values are as follows: 1.Individuals and Interactions Over Processes and Tools; 2. Working Software Over Comprehensive Documentation; 3. Customer Collaboration Over Contract Negotiation, and 4. Responding to Change Over Following a Plan.

The Agile Manifesto [3] aimed to realign and reprioritize the importance of the different moving parts in the SDLC, giving lenience towards following processes, tools, comprehensive documentation, contracts, and sticking to a defined plan as long as the changes aided the mission of providing software that successfully met the requirements in a timely manner.

With these four values in mind, the SDLC process was modified. Instead of the costly process of going through the development phases sequentially, the Agile Methodology encouraged more incremental, iterative, and simultaneous development of the SDLC phases. This type of approach allowed for more flexibility to adapt to changes in requirements, overcoming hurdles, and lowering the cost of doing so. This iterative and incremental development approach came a long way from the traditional Waterfall Model's measurement of progress in a linear sequential process.

Agile teams lean on automation tooling that help enable continuous integration and continuous delivery (CI/CD). We will look more at CI/CD later on.

While the Agile Methodology solved some of the problems that were defined in the Waterfall Model, it was discovered that there was something that the Agile Methodology did not account for. Once the software was developed, tested, iterated on, and deployed—what was next? There was another facet to the SDLC that was not focused on: operations and maintenance. This refers to the cycle of keeping the software running, identifying improvements that need to be made (through continuous monitoring and observability features), and implementing those improvements. Enter: DevOps.

2.1.3 Agile and DevOps. Just as the drawbacks of the Waterfall Model inspired the Agile Methodology, the drawbacks of the Agile Methodology similarly inspired the DevOps movement. Although often used interchangeably, these methodologies are not the same [19]. The additional factor that DevOps brings is explained by its name. DevOps refers to the merging of development operations and IT operations.

Before the DevOps movement started, the IT/operations folks were the ones who focused on monitoring and maintaining a variety of things like uptime, security, network, integrations, reliability, and compliance. These functions require strong communication and cross-functional testing when it comes to introducing code changes or new features to an existing software product/system to ensure that the integrity, functionality, and scalability of the product does not become compromised.

The lack of this collaboration between development teams and operational teams is what resulted in various unintended consequences (the aforementioned drawbacks of Agile): creating separate software components that could not integrate with one another,

were not extendable, made breaking changes to the existing software, etc. These things could have been avoided mostly (if not entirely) by having an even tighter feedback loop and collaborative structure set up between the development and operational teams.

DevOps is not necessarily a replacement for Agile, but rather another framework that works in conjunction with the Agile methodology. DevOps may compromise some of the speed, leanness, and overall "agility" of Agile but all while introducing smoother cross-functional work, increasing the frequency and quality of communication, holding automation to a higher value, and maximizing efficiency as much as possible (Watts, 2017). Adding DevOps to the equation requires cultural shifts, emphasizing cross-functional collaboration and roles, and increase overall responsiveness in the development process [17].

2.1.4 DevSecOps and the SSDLC. With the industry focus shifting from fast development and deployment to now considering security implications and wanting to prevent exploitation of security vulnerabilities, much research has been done to start formally introducing cyber security into the SDLC.

The idea was to introduce security earlier in the SDLC. Adding security engagement at the end of the cycle proved to be difficult because all of the major infrastructural and design decisions had already been made. If any major security concerns were identified, it became costly and difficult to make these changes prior to release.

This idea of "shifting security left" in the SDLC prompted creation of the Secure Software Development Lifecycle (SSDLC) or the Secure Development Lifecycle (SDL).

Along with this new process definition also came the evolution of DevOps into DevSecOps. This refers to the injection of security principles and controls into the DevOps model that again integrates development and operational work while focusing on cultural shifts to empower the success of the new development process.

While the advent of DevSecOps has solved the problems derived from the lack of an inclusive feedback loop between developers, security engineers, and IT/operations, successful adoption of DevSecOps into the enterprise technology world is slow and ongoing. The DevSecOps model requires both cultural and infrastructural changes across multiple cross-functional organizations within a large corporation. These same obstacles foreshadow some of the challenges we are sure to face when introducing UX design principles into the DevSecOps/SDL model. We will discuss these challenges in further detail later on in this paper as we discuss the proposed solution for merging the gap between UX design and security.

2.2 UX Research and Design

Before jumping into merging the gap between UX and security, we need to explore the context of UX design, UX research, and why these are important.

UX designers are responsible for making a product useful, effective, and intuitive for its users. The duties of a UX designer may vary depending on the project—they may conduct user and product research, create personas for the target users of a product, define the information architecture for the product, start mocking up wireframes for the designs of the product UI, turning the wireframes into prototypes by adding interaction choices (e.g. animations, navigation options, etc.), and finally, usability testing.

UX research is more specifically focused on understanding effective design choices, designing for accessibility, user behaviors, user goals/needs/pain points, etc. This may include interviewing users and stakeholders, conducting surveys, hosting focus groups, or conducting a competitive analysis with similar products in the market [1].

UX research and design are both extremely important in determining the success of a software product. If you have a functional product that is extremely difficult to use, then your user base will decline and suffer. Adding to the decrease in effective usability and design: the creation of limiting/restrictive features in a software product. Specifically, security controls. This is where the notion of "usability versus security" comes into play.

2.3 Usability and Security

The study of UX has evolved and researchers have started looking into how we can maintain a software's usability within the development lifecycle. Some examples or how usability and security can conflict include: methods of multi-factor authentication, password policies, and encryption.

Users have complained about having to enable two-factor authentication (2FA) or multi-factor authentication (MFA) in order to login to certain accounts. To quote a lawsuit against Apple for forcing 2FA on Apple accounts, "millions of...consumers across the nation have been and continue to suffer harm...in terms of the interference with the use of their personal devices and waste of their personal time in using additional time for simple logging in" [18]. Although 2FA adds additional verification security to user accounts, the process must be designed in a path of least resistance to the users.

Similarly, password policies have been known as a pain point to new user registration flows on certain platforms. Password policies that have too many restrictions in order to supposedly ensure that the user's password is strong can prove to be frustrating for someone trying to create a new account [6]. Sometimes companies may believe that more restrictions on creating a password result in stronger passwords, but that's not always the case. Password policies like a maximum character length, required use of specific numbers of special characters or non-consecutive numbers may not always result in the highest entropy of a password. With too many failed attempts at creating an "acceptable" password, this decline in usability may impact a platform's user growth.

Encryption, a way of protecting data from unauthorized parties, is another important security measure. Pretty Good Privacy (PGP) is a specific encryption program that can be used to sign, encrypt, or decrypt data. Although PGP is (maybe one of the most) well-known technologies used for encrypted communications, the usability of the software is so low that its inventor—Phil Zimmermann—doesn't even use it [8].

Security by design/default is something that is often proposed as a solution to solving this problem. Security by design implies that security engineers are involved in the design of a product. By doing this, the intent is to secure the product and have secure defaults/failover modes to further protect the users [11]. Others go

further to say that we "simply" need to include security in both the software design and UX design phases of the development process with the ideal state of UX designers becoming educated in "the basics of security and authentication" [10].

This is easier said than done. People who work in software, security, and design specialize in their respective fields. It is hard to give each of these individual contributors enough resources, training, and time to learn the ins and outs of all three fields—even if this were only limited to the basics. Much of the literature suggests that we need to do more research in this area, including systemic mapping studies and literature reviews to fully understand how we can approach the solution to the competing factors of UX design and agile development as well as security [12]. The key take-away from the current work in this area would reinforce the fact that UX designers need to be looped into the SDL, thus prompting another infrastructural and cultural shift from the idea of "usability versus security" to "usability and security". This will serve to address the challenges faced by both home and organizational users alike [4, 5]. The proposed solution in this paper aims to provide the actual details of how such a thing can be implemented in order to bridge this gap between design, software, and security.

3 A NEW APPROACH

3.1 Agile, SSDLC, and DevSecOps Models Are Not Enough

The UX missing from the DevSecOps workflow has not been a large focus area. The focal point of DevSecOps until now been formalizing and tightening the feedback loop between software and security. This has been researched and proposed as the move from DevOps [2] to DevSecOps and the SSDLC/SDL [16].

It only makes sense that the introduction of another party into the feedback loop be tackled in a way similar to the movement from DevOps to DevSecOps.

3.2 Adapting the SSDLC into the TDLC

What we are proposing here is another adaptation of the SDL and the DevSecOps model for agile development. We want to include the UX designers and researchers in the formalized model that we are proposing as the Technology Development Lifecycle (TDLC).

This section of the paper will focus on identifying how the UX design and research processes will be looped into the DevSecOps model and designing a CI/CD framework (an automated tool suite for development) to enforce the TDLC.

3.2.1 The Double Diamond Model. We have already gone over the current DevSecOps model that illustrates a feedback loop between software, security, and IT/operations. Let's take a look at the current UX design workflow.

The Double Diamond model is a widely used model in the UX design process that was developed by the British Design Council in 2005. The four key steps laid out in this model are: 1. Discover; 2. Define; 3. Develop, and 4. Deliver.

During the "Discover" phase, UX research is conducted to scope out project and its problem space. This research is then analyzed and used to synthesize a list of pain points that need to be considered in the design of the product. This is used in the "Define" phase to fully define the specific problems, what needs to be improved, and what the proposed solutions are. The "Develop" phase is where UX designers begin creating sketches and wireframes for the product design/redesign. They deliver these as prototypes [15].

The left half of each diamond represents divergent thinking, and the right half of each diamond represents convergent thinking. A designer would want to practice divergent thinking when performing things like user/product/market research or during the ideation of designs and wireframes. Convergent thinking is critical after each of the divergent phases because what has been explored needs to be pared down and focused. If a designer has followed the Double Diamond model properly, they would have started out the process with a general idea of the problem statement, defined specific problem areas to tackle in their design phase, and come out of the process with specific solutions.

3.2.2 TDLC Model. The goal of the TDLC is to merge the DevSecOps model with the Double Diamond Model. This will help us include the UX design process into a DevSecOps agile development model. The TDLC Model follows the same cyclical style of the DevSecOps model to show the iterative manner of this process. The TDLC introduces one more circle into the infinity loop where the four phases of the Double Diamond model are included at the beginning. The design phases flow into the development phases, then into the operations phases, then back to the start. All three parts of the new infinity ring are encompassed by security to represent the inclusion of security controls and tools throughout the process.

3.2.3 TDLC Workflow. There are a few different parts of the TDLC workflow. We will break them down into digestible sections to better illustrate each flow.

Version Control For All: Design, Plan, and Document It is important to maintain full visibility between all parties within the TDLC. In order to do this, we would like to introduce the use of a version control system to the designers in the workflow.

When checking files into the version control system, things should be structured in a standardized way in order to keep things organized. If the same format is always followed for the project structure, then the cross-functional teams will be able to find what they need every time. This will also make things easier for any automation later on.

Design, Planning, and Documentation The TDLC workflow starts off with the UX designers in the research phase. They use their collected research to define the specific problems, scope of the problems, propose effective design solutions. All of this will be recorded in documents that must be peer-reviewed by design peers, the tech lead for relevant the software engineering team, and the security champion designated for that project.

The design documents will be committed to the shared version control repository. The designers will then work on iterating their actual designs and prototypes. These prototypes are passed over to the development team(s) with access being provided the security team as well. To ensure that the security team sees all of the designs for security-related UI features, these can be tracked with an additional metadata field in an issue tracking system.

Once the design prototypes are delivered to the engineering teams, the engineers start scoping out the formal requirements of the software itself. They will create the requirements document,

design the architecture, and write a technical design document (TDD) during this phase. They will work alongside the security engineering team who will be aiding with security consulting, security requirements, threat modeling, compliance, and reviewing the TDD for potential security concerns. These technical documents will then also be committed to the shared version control repository.

Continuous Integration There will be a CI system that will help facilitate the workflow of events for design, development, security, and operations. More on this in the next section.

Continuous Development The code will move forward into a code review cycle. If the revision is declined, then the developer will have to go back to fix and iterate on the issue found. If the revision is accepted, this prompts the merge of the code into the master project.

Deployment Environments Once everything is good to go, the project can be deployed. There are three different environments that a project can be deployed to. There is the development environment (which is used by developers when they are building the project), the staging environment (where testing happens e.g. dynamic code analysis, stress testing for performance, penetration testing, usability testing), and of course the production environment (used for final release to the intended customer of the product.) If any issues are found during these tests, an issue is created and the designers/developers will be returned to another iteration of their design/code.

Continuous Monitoring In order to make sure the software remains running properly, there needs to be a system in place for continuous monitoring. This will ideally alert the appropriate parties if any issues come up in terms of availability, security, functionality, and so on. Any issues found will be tracked and added to the backlog of things to be improved on the next iteration.

3.2.4 Implementing the TDLC as a CI/CD Working Environment. Since we would like to actually implement a CI/CD working environment to enable the use of this new TDLC model, let's take a look at some of the logistics and workflows.

As mentioned, it will be important to unify all parties by encouraging the use of the shared version control system. Figure 1 illustrates this workflow.

The overall workflow would be based on using the version control system to track any changes. When changes are submitted, then the user (designer, developer, or security) will be moved through the CI processes, CD processes, and then eventually into the operational tasks until they loop back to the beginning of the flow.

Two of the main use cases to highlight the workflow are:

Designer: 1. Design Commit; 2. Design Review; 3. Usability Testing, and 4. Redesign AND Iterate OR Deliver to Developers

Developer: 1. Code Commit; 2. Build; 3. Testing: QA, Static Code Analysis, Unit Testing, Integration Testing, and 4. Create Issue OR Pass/Dismiss

In order to successfully implement this CI/CD working environment for the TDLC, we have put some thought into what technologies and frameworks could be leveraged.

3.3 Version Control System

We have decided that git would be the best option for version control, as this has become an industry-wide standard. Subversion (svn) is another option, although it is being used less and less in the tech industry. For this CI/CD workflow, we could leverage a tool suite like GitLab which supports git and offers many CI/CD features as well.

For designers, we would want to integrate something like Versions by Sympli. This is a UI that integrates with git specifically for managing versions of graphic and wireframe/prototype files. This tool displays the differences made between each version of the design files and supports many plugins with commonly used design tools. There are multiple ways to interact with git and the most common way to do so is via the command-line, but for the designers that haven't had experience with this, this tool will make the version control system more accessible in this new workflow.

3.4 Design Tools

For the initial implementation we would like to support these commonly used tools: 1.Adobe Illustrator; 2. Adobe XD, and 3. Sketch.

3.5 Issue Tracking System

An standard tool used for issue tracking is JIRA by Atlassian. We would use this for creating issue tickets to track any jobs-to-be-done for the involved parties in the TDLC.

3.6 CI/CD Server and System

As mentioned, GitLab supports automation of many CI/CD tasks. GitLab offers GitLab Runners for automated build management. They also offer automated testing features, deployment, and ongoing monitoring for operational software after deployment.

3.7 CI/CD Alternatives

If we don't end up using GitLab for every aspect of CI/CD automation, some alternative technologies to consider include Apache Ant (Build), Selenium (Test), and Jenkins, Puppet, or Chef (Deploy).

4 CHALLENGES

It was (and still is) already a challenge to successfully implement the DevSecOps model to bridge the gap between software and security. Software and security are two fields that entail their own workloads and priorities—this makes it difficult to prioritize cross-functional work.

The underlying issue here is that this shift requires more than just redesigning the SDLC. There are many moving factors including cultural changes, incentivizing cross-functional work and milestones, and overall understanding that all the shared work is important [9]. It will take more time and more support from upper-level management/leadership in software companies to really enforce this as something that needs to be adopted widely across organizations. Adding another element here is a challenge because we haven't yet properly implemented the DevSecOps SDL movement.

5 FUTURE WORK

The next steps would be to continue testing/reiterating on this new model prior to the actual implementation of the CI/CD working environment for the TDLC. Many of these types CI/CD processes are already set up and automated at enterprise level for technology

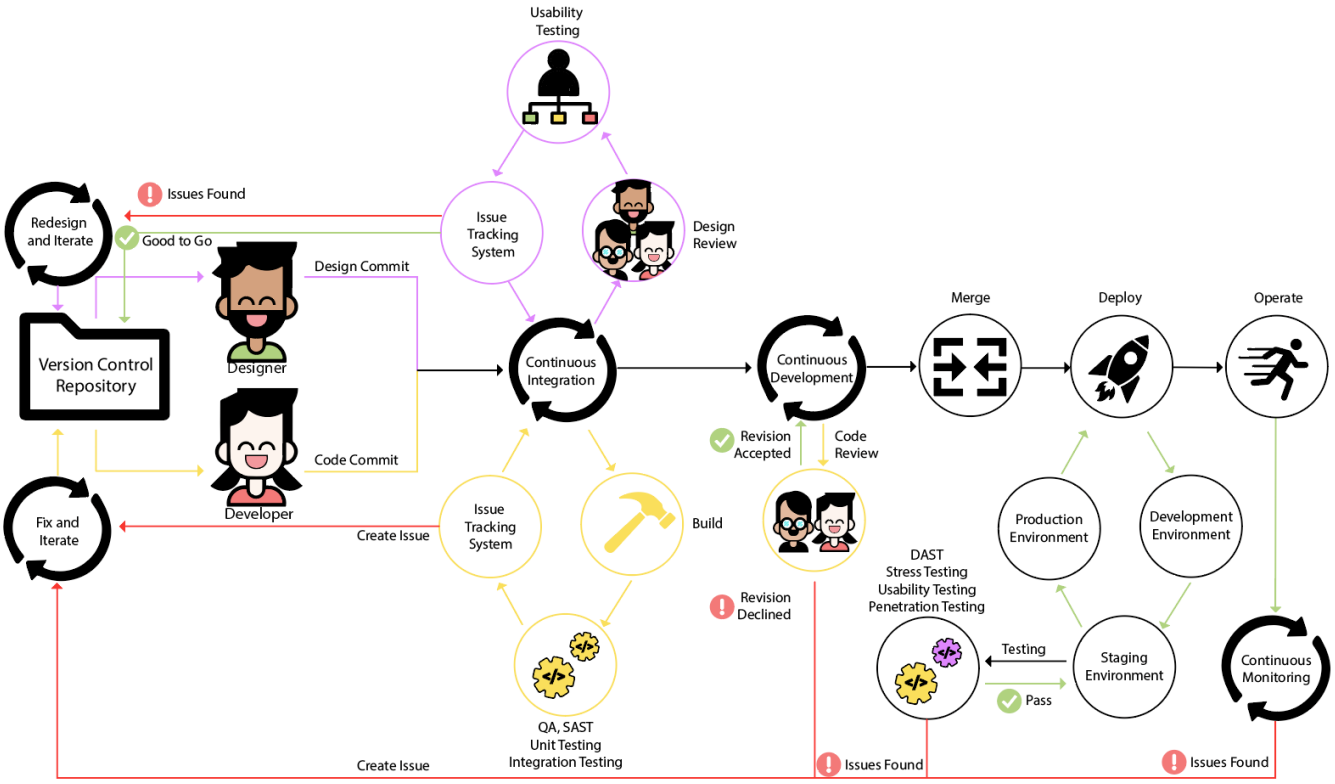

Figure 1: CI/CD Workflows © Jessica Nguyen

companies. In implementation, we'd want to stick with industry standard technologies to allow for ease of transition into the TDLC.

REFERENCES

[1] Nick Babich. 2017. What Does a UX Designer Actually Do? https://theblog.adobe.com/what-does-a-ux-designer-actually-do/

[2] Soon Bang, Sam Chung, Young Choh, and Marc Dupuis. 2013. A Grounded Theory Analysis of Modern Web Applications: Knowledge, Skills, and Abilities for DevOps. In *Conference on Research in Information Technology*. ACM, 61âĂŞ62. https://doi.org/10.1145/2512209.2512229

[3] Kent Beck, Mike Beedle, Arie Van Bennekum, Alistair Cockburn, Ward Cunningham, Martin Fowler, James Grenning, Jim Highsmith, Andrew Hunt, and Ron Jeffries. 2001. Manifesto for agile software development. (2001).

[4] Marc Dupuis and Robert Crossler. 2019. The Compromise of One's Personal Information: Trait Affect as an Antecedent in Explaining the Behavior of Individuals. In *Proceedings of the 52nd Hawaii International Conference on System Sciences*. IEEE, 4841–4850. https://doi.org/10.24251/HICSS.2019.584

[5] Marc Dupuis and Samreen Khadeer. 2016. Curiosity Killed the Organization: A Psychological Comparison between Malicious and Non-Malicious Insiders and the Insider Threat. In *Proceedings of the 5th Annual Conference on Research in Information Technology*. ACM Press, 35–40. https://doi.org/10.1145/2978178.2978185

[6] Marc Dupuis and Faisal Khan. 2018. Effects of peer feedback on password strength. In *2018 APWG Symposium on Electronic Crime Research (eCrime)*. IEEE, 1–9. https://doi.org/10.1109/ECRIME.2018.8376210

[7] Geoffrey Elliott. 2004. *Global business information technology: an integrated systems approach*. Pearson Education.

[8] Lorenzo Franceschi-Bicchierai. 2015. Even the Inventor of PGP Doesn't Use PGP. https://www.vice.com/en_us/article/vvbw9a/even-the-inventor-of-pgp-doesnt-use-pgp

[9] Patricia Johnson. 2017. Shifting Security Left: 3 DevSecOps Challenges and How to Overcome Them - DZone DevOps. https://dzone.com/articles/shifting-security-left-3-devsecops-challenges-amp

[10] Nicole Kobie. 2016. Security vs usability: it doesn't have to be a trade-off. *The Telegraph* (Jul 2016). https://www.telegraph.co.uk/connect/better-business/security-versus-usability-ux-debate/

[11] Monique Magalhaes. 2018. Security vs. usability: Does there have to be a compromise? http://techgenix.com/security-vs-usability/

[12] Daniel A. MagÃijes, John W. Castro, and Silvia T. Acuna. 2016. HCI usability techniques in agile development. In *2016 IEEE International Conference on Automatica (ICA-ACCA)*. IEEE, 1–7.

[13] Ernest Mougoue. 2016. What is the secure software development life cycle (SDLC)? | Synopsys. https://www.synopsys.com/blogs/software-security/secure-sdlc/

[14] Manzoor Ahmad Rather and Mr Vivek Bhatnagar. 2015. A Comparative Study of Software Development Life Cycle Models. *International Journal of Application or Innovation in Engineering & Management (IJAIEM)* 4, 10 (2015), 23–29.

[15] Jonny Schneider. 2015. The Double Diamond: Strategy + Execution of the Right Solution. https://www.thoughtworks.com/insights/blog/double-diamond

[16] Dave Shackleford. 2017. *The DevSecOps Approach to Securing Your Code and Your Cloud*.

[17] John Steven. 2018. What's the difference between agile, CI/CD, and DevOps? | Synopsys. https://www.synopsys.com/blogs/software-security/agile-cicd-devops-glossary/

[18] Lisa Vaas. 2019. Apple sued for 'forcing' 2FA on accounts. https://nakedsecurity.sophos.com/2019/02/12/apple-sued-for-forcing-2fa-on-accounts/

[19] Stephen Watts and Chrissy Kidd. 2017. DevOps vs Agile: What's the Difference and How Are They Related? - BMC Blogs. https://www.bmc.com/blogs/devops-vs-agile-whats-the-difference-and-how-are-they-related/

[20] Ron B. Yeh, Andreas Paepcke, and Scott R. Klemmer. 2008. Iterative design and evaluation of an event architecture for pen-and-paper interfaces. In *Proceedings of the 21st annual ACM symposium on User interface software and technology*. ACM, 111–120.