SECURITY IN AGILE DEVELOPMENT, USE CASE IN TYPEFORM

Pau Julià, David Salvador and Marc Peña

Security Team, Typeform, Barcelona, Spain

ABSTRACT

Software development methodologies have evolved during the last years to reduce the time to market to the minimum possible. Agile is one of the most common and used methodologies for rapid application development. As the agile manifesto defines in its 12 principles, one of its main goals is to satisfy the customer needs through early and continuous delivery of valuable software. Significantly, that none of the principles refers to security. In this paper, we will explain how Typeform integrates security activities into the whole development process, reducing at the same time the phases on the S-SDLC to reduce friction and improve delivery while maintaining the security level.

KEYWORDS

Security, S-SDLC, SDLC, AGILE, Development, Methodology, Use Case.

1. INTRODUCTION

Software development methodologies have evolved during the last years to reduce the time to market to the minimum possible. Methodologies like waterfall cannot adapt to these goals, so new methodologies arose to gain flexibility and allow rapid development, commonly called RAD [1] (Rapid Application Development), but at the same time unwittingly pushing the security checks and reviews out of the development and delivery flow. Agile [2] is one of the most common and used methodologies for rapid application development. As the agile manifesto defines in its 12 principles, one of its main goals is to satisfy the customer needs through early and continuous delivery of valuable software. Significantly, none of the principles refers to security.

2. CURRENT CONTEXT

In a waterfall development model, organizations usually performed security-related activities during the non-functional testing phase [3]. During this phase security engineers reviewed, tested, and analyzed the implemented solution, or new release of the software, before going live to production. This situation contributed to the creation of friction between the security and the engineering teams. This friction was a consequence of the security team's output during that testing phase: bugs to be fixed, new requirements, etc. Basically adding more workload to the development team after their implementation phase. This new workload also meant a blocking situation for the implementation to go live, affecting business goals and needs. On top of that, the security team's perception of the organization could be affected negatively by being seen as a "blocker" rather than helping the organization.

David C. Wyld et al. (Eds): MLDS, NECO, SEMIT, IBCOM, SPPR, SCAI, CSIA, ICCSEA - 2021
pp. 99-109, 2021. CS & IT - CSCP 2021 DOI: 10.5121/csit.2021.111809

Most organizations had, and still have, a low ratio between security team members and engineering team members. This situation adds extra complexity to the task of improving security organization-wide. On many occasions, the security team will be the bottleneck of the organization. A simple way of trying to solve this problem would be to add a security team member to every engineering team. However, most organizations will not be able to do so due to costs, resources, and scalability reasons. Nowadays, the ratio between security and engineering headcount in Typeform is 1 to 30.

2.1. From Blockers to Facilitators

Security bugs will be found when the security team tests the implemented software. Those security bugs are added to the engineering team's backlog which may also include functional, UI, performance, and other types of bugs. This addition of more work to the team's backlog might cause fatigue and disengagement.

Unsurprisingly the security team has been perceived as the "NO department"; the gatekeeper that decides what can and what cannot be done with the information and with the technology; the team that blocks projects, tasks, ideas and introduces a delay in the delivery [4]. As a side effect of this, interaction with the security team is avoided as much as possible and not considered even for consultation in companies where security is not the core of the business.

Certain security policies put in place by the security team due to compliance or legal requirements tend to forbid or limit the usage of tools, libraries, and other software. In addition to that, certain security policies might force the organization members to go through slow and exhaustive processes to reach certain ends. This kind of situations push organization members to find ways to avoid those policies so, instead of promoting security within the organization, they lower it.

The goal and the mission of the security department must be just the opposite: identify critical risks affecting the critical assets and treat them with effective controls and be flexible with the less important ones.

Reducing friction with the rest of the teams and becoming an active facilitator team, will help improve the security across the company. On top of that, engineering teams will see security as a status to achieve and not as an obstacle. The only way to achieve that is to refocus security in a way that is non-intrusive yet effective.

2.2. Shifting Security to the Left

Systems Sciences Institute at IBM reported [5] that it costs six times more to fix a bug found during implementation than one identified during design. In addition, it is reported that the cost to fix bugs found during the testing phase could be 15 times more than the cost of fixing those found during design.

A paradigm called "shift security to the left" [6] was created aimed to identify security bugs at the earliest development stage possible. One of the key principles is that the responsibility of security should be shared with the engineering team. For this "shift" to be successful the security team must provide efficient tools and processes to the engineering team that will allow them to be as productive as before with the least amount of friction. For example, providing an IDE plugin to detect vulnerable dependencies as they are added.

As new tools and processes are provided to the engineering team new frictions in the development workflow can appear. For example, in an organization where every committed code is scanned for third-party vulnerabilities, giving the full information of the vulnerable component to the engineer can block the development. These details are useful for the security team but they can be too domain-specific. Following the previous example, a better way would be to inform the engineer that a dependency should be upgraded and provide an easy way to do it. The role of the security team is to provide tools and processes that are developer-friendly.

3. S-SDLC AT TYPEFORM

In this section, we explain how Typeform implemented an ad hoc Secure Software Development Life Cycle (S-SDLC) [7], its benefits, challenges, and problems found during its implementation. The S-SDLC described in this section was designed having in mind an organization with an agile development methodology and a short time to market need.

Typeform is an online Software-as-a-Service (SaaS) company that specialises in online form building and online surveys. Its main software creates dynamic forms based on user needs. Due to that, Typeform is dealing with a huge volume of information from its clients and the respondents of their forms and this implies a big responsibility to guarantee the confidentiality, integrity and availability of that information.

At a software engineering level, it follows an event-based microservice architecture approach. Every microservice is owned by one of the 20 engineering teams that exist in Typeform. The average deployment rate of a new version of a microservice is 0,5 hours (around 50 per day).

From a security perspective, an organization that continuously deploys new versions of its microservices is a double-edged sword. On the bright side, the same agility that allows new features to be fastly released, allows for security fixes to find their way into production in a quick way. So, attack windows will be shorter than in other less agile organizations. As a drawback, it is not feasible for the security team to keep track of every change or addition made to the production environment. Instead, the security team should provide tools, processes, and controls to avoid bringing flawed implementations into production.

3.1. Goals

Typeform's S-SDLC has been designed to achieve the following goals:

Goal 1 – Developments securely achieve requirements and objectives from design.
Goal 2 – Security risks are identified and treated as soon as possible, ideally before they appear.
Goal 3 – The S-SDLC activities have a minimal impact on the development's time to market.
Goal 4 – Security team's perception is improved throughout the organization.

3.2. Phases

Given Typeform's SDLC, agile and with a high rate of deployments, we classified the security related activities added to the SDLC in three steps: build, deploy, and run (Figure 1).

This classification follows an approach similar to a DevOps culture one: multidisciplinary teams with a high degree of autonomy and accountability of their developments. These teams prioritize mean time to mitigate (MTTM) and mean time to remediate (MTTR). Therefore, the workflow of

these teams is heavily based on deploying code to production in a fast way. To not be disruptive to the organization, the S-SDLC has to be built around keeping these teams' objectives.

We consider as an S-SDLC activity any task or process, automated or not, that evaluates the security of the implementation and produces an output. This output can be, for example, in the form of a code fix, a design recommendation, or a vulnerability report. Next, we will describe every S-SDLC activity added in every phase.

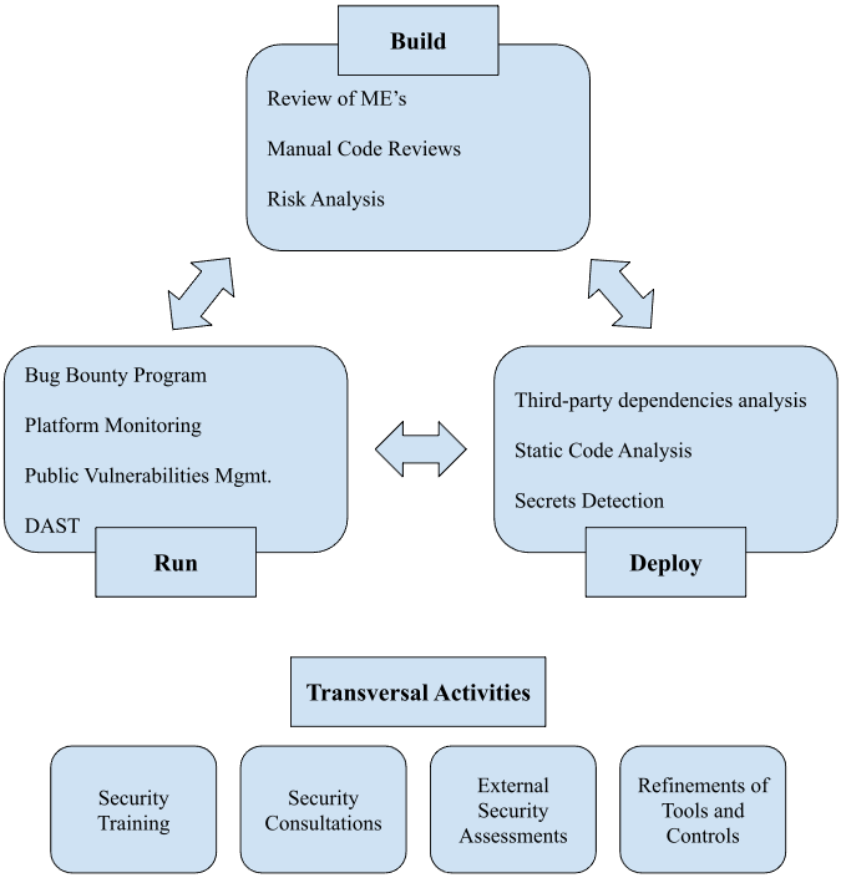

Figure 1. S-SDLC Phases and their security activities

3.2.1. Build

We consider this phase to start when a new development is conceived and to finish when the first implementation of it is done. That includes designs, spikes, or any other activity performed before there is an implementation complete enough to be deployed to production. Functionality is the factor that marks the suitability for its deployment in production. It also includes code that is deployed to testing environments.

The S-SDLC activities included in this phase are:

3.2.1.1. Review of Meaningful Experiments

In Typeform, a Meaningful Experiment (ME) is any A/B test [8], feature or technical work implemented in the product application. This concept has been created to empower engineering

teams to experiment, learn from the outcomes, and consistently deliver quality and improvements.

Every ME is started by one or multiple engineering team members creating a Wiki-style page with definitions, potential problems, architecture changes, and any other useful information. The ME is then reviewed by different organization stakeholders: data analytics team, infrastructure team, architecture among others.

The security team is also one of the stakeholders who will review the ME. Before the review process, the authors of the ME have to fill out a small questionnaire with security questions. The answers will help the security team have an overview of what will be modified and how. The engineering team can start working on the ME implementation but before moving to the next phase, deploy, the ME needs to be approved by all stakeholders, including the security team. If appropriate, the ME can be labeled as "Security Review" meaning that further actions need to be taken by the security team. Those can be manually reviewing the source code, usually in the form of a Pull Request, dynamically testing the implementation when it reaches production or any other follow-up activity during the implementation's lifecycle.

Leveraging the MEs the security team is aware of every new implementation and can detect security problems at a very early stage. Given that the MEs reviews are made asynchronously, the security team will not be blocking their development. For the cases where new requirements are arisen by the security team, those will not create a heavy workload on the engineering team as they are still in the early development stages.

As a sample we took a period of 12 weeks, where 384 MEs were created (Figure 2); an average of 32 per week. Most of those MEs had no security relevance, for example, a ME about a color scheme redesign. From the MEs which had security relevance and needed further analysis, 49 of them were canceled during the Build phase, 43 of them needed further information, design, and documentation work, and 84 of them were accepted to move forward. Finally, from the MEs moving forward only 13 of them, a 3% of the total, were tagged for further Security Review.

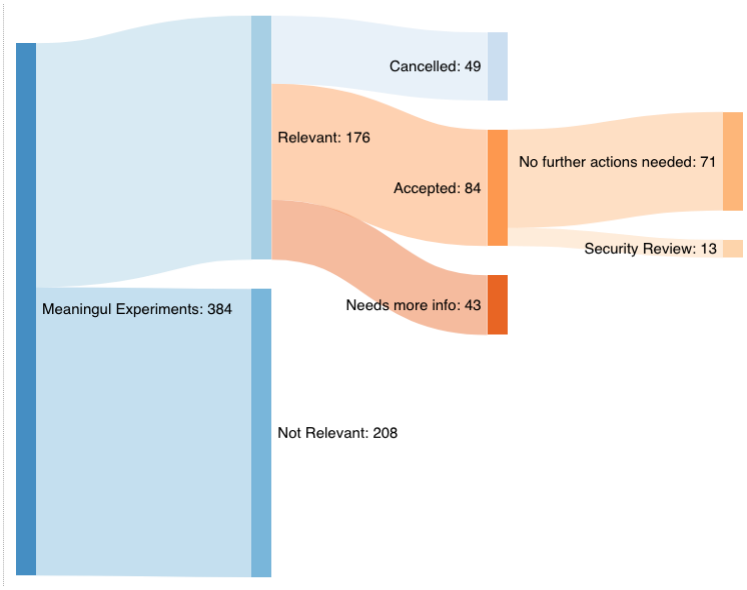

Figure 2. Meaningful Experiments review for 12 weeks.

3.2.1.2. Manual Code Reviews

Certain implementations related to sensitive features require a code revision by the security team: some examples are authentication and authorization, changes to core architecture, integrations with third parties, user-provided webhooks, etc. Those reviews are usually performed through one, or multiple, Pull Requests where the security team is added as a mandatory reviewer. As previously mentioned, this situation happens when a ME has been labeled as "Security Review". This situation allows the security team to have a good overview of sensitive features and their modifications over time. However, we found that in some cases this can be blocked: it is not until a security member has reviewed the code that the implementation can move forward. This blocking situation depends on the workload of the security team and its prioritization.

3.2.1.3. Risk Analysis

As with Manual Code Reviews, certain MEs labeled as "Security Review" will go through extra analysis during the Build phase. In this case, an ad hoc risk analysis will be performed by the security team.

The objective of this risk analysis is to evaluate the effects that the requirements defined in the ME will have against the organization's security in terms of information confidentiality, integrity, availability, and auditability. The outcome of the analysis will come in the form of action to be taken to accept, mitigate, avoid or reject the identified risks.

The risk analysis also includes a threat modeling process. This process aims to: identify security objectives, break down the application's architecture to identify potential security attacks, identify and categorize potential security attacks, and estimate the risk.

3.2.2. Deploy

This is a short-timed but crucial phase. This one includes the time range between a code implementation is ready to be deployed to production to when it is finally live in production.

All of the activities in this phase fall under the category of Static Application Security Testing (SAST), where the source code is scanned for security issues. The output of these scans is then used to decide if the implementation is moving forward and being deployed to production or if it needs to be fixed.

These activities reinforce the "Shifting Security to the Left" paradigm by providing scan results and fix suggestions in an early phase where the code has not reached production.

The S-SDLC activities included in this phase are:

3.2.2.1. Third-party dependencies vulnerability analysis

The third-party dependencies included in every source code repository are monitored automatically by a tool with a vulnerability database that is continuously updated.

When a vulnerability is detected in one of the third-party dependencies, and a fix exists, a Pull Request is created in the affected repositories. This Pull Request includes a fix for that vulnerability which is usually a version upgrade. The Pull Request also includes an automatically generated digest of the vulnerability and its severity. This flow helps the security team become a facilitator, as a solution is already being provided (the fix Pull Request) instead of just opening an

issue in the organization's ticketing system, reducing the friction between the engineering and the security team.

The analysis of third-party dependencies is also made on every Pull Request made to the repository by the engineering team. The output of the scan will inform the Pull Request author whether a component with vulnerabilities is being introduced through that Pull Request or not. This flow helps to shift security to the left as the engineering team is responsible for that Pull Request.

3.2.2.2. Static Code Analysis

The most critical and sensitive source code repositories are integrated with a commercial Static Code Analysis tool.

For every new commit made to the main branch of the repository, a package with the new code is generated and sent to the Static Code Analysis tool. The tool scans the new code for vulnerabilities and stores the results. Afterward, a notification is sent to the security team if new vulnerabilities have been detected. If so, a security team member reviews the results and, if appropriate, opens an issue in the organization's ticketing system describing the vulnerability and suggesting a fix.

The Static Code Analysis scan result does not block any development, it is an asynchronous process that keeps an updated vulnerability status of the source code. The development can proceed and the security team reviews the results when convenient or needed.
It's been challenging to find a balance when choosing the number of repositories to scan and not introduce a lot of noise by notifications and excessive workload.

3.2.2.3. Secrets Detection

Unfortunately, uploading by mistake credentials, tokens, keys, or any other type of secret to the source code repository is a common issue. The impact of these incidents can be fatal, especially if the repository is publicly available; an attacker can use those leaked secrets as a starting point for a much bigger attack such as data exfiltration, defacement, advanced persistent threat, etc.

To avoid this we included what we call a "secrets detection" control in every Pull Request and commit made to the source code repository. The added code is scanned looking for patterns that might match with tokens, credentials, or any other type of secret. If any of the patterns match the engineering team member submitting that code is notified and redirected to a Wiki-style page with the steps to follow: revoke the secret and remove it from the source control versioning system.

In the early versions of our secrets detection control, the detection patterns needed several tuning as the rate of false positives was high. After some iterations, we managed to lower that rate while still covering a wide range of potential cases.

3.2.3. Run

This phase can be better described as a status. It refers to the moment where the code is live in production and remains there. However, that running code might be available only to a part of the users through the usage of "Feature Flags": a software practice of gating functionalities. A functionality, or feature, can be turned on and off via a management service allowing certain users, for example, the ones geographically located in Europe, to have access to that feature.

The S-SDLC activities included in this phase are:

3.2.3.1. Bug Bounty Program

Bug Bounty programs adoption is becoming very spread among organizations. This type of program offers recognition and usually an economical compensation to individuals who report bugs of certain software or service, especially bugs related to security vulnerabilities.

Every bug reported to the program is first triaged by an external security team part of the bug bounty program. If the bug is confirmed then Typeform's security team comes into action by manually reviewing the issue and triaging it. This manual intervention depends on the bug's severity and impact. Depending on those factors a further investigation to identify the root cause and if that same bug exists somewhere else in our product is performed. On multiple occasions we have encountered bugs that existed in different parts of the code too and, thanks to the internal expertise of our security team, we were able to identify them before being reported again in the Bug Bounty program.

Having a Bug Bounty program in place helped to close the gap between the amount of engineering team members and security team members. This gap is closed with the addition of both the individuals reporting bugs to the program and the external security team performing a first triage of the bugs.

Managing a Bug Bounty program has its challenges too; triaging timelines need to be met which can result sometimes in excessive workload to a small security team. Plus, certain disagreements with the bug, its severity, and impact sometimes happen between the reporting individual and both the external and Typeform's security team.

3.2.3.2. Platform Monitoring

It is key to our platform, the SaaS offered by Typeform, to have a security monitoring program. This is achieved by collecting all the data produced by the infrastructure, applications, and tools that power our platform. Once collected it can be analyzed and exploited allowing the security team to identify in early stages any suspicious activity or potential security vulnerability. This early identification helps to block those suspicious activities potentially before harm can be done. Other benefits include audit compliance, service level functional monitoring, performance measuring, liability limitation, and capacity planning.

Given the dimension of our platform, a fixed number of security team members need to be assigned to this task.

3.2.3.3. Public Vulnerabilities Management

Typeform's platform infrastructure is continuously scanned for public vulnerabilities affecting any of the components such as operating systems, virtualization software, networking components, etc. The scan is performed through a set of tools and services that inform the security team whenever a vulnerability is detected in any of the monitored components.

As with the Platform Monitoring activity, a fixed number of security team members need to be assigned to the management and refinement of this activity. Usually the same members given the shared knowledge and expertise on those matters.

3.2.3.4. Dynamic Application Security Testing (DAST)

Once the code has reached production and is running, it is tested for common web application vulnerabilities. This is done through a commercial software tool that tests for injection flaws, like SQL injection and Cross-Site Scripting, broken authentication, business logic flaws and other types of web application vulnerabilities. These tests are performed periodically and the output of the tests is sent to the security team for further review if needed.

Whenever a Meaningful Experiment has been labelled as “Security Review” these types of tests will also be carried out by a security team member manually and tool-assisted. To reduce the risk of having sensitive untested code running in production we use “Feature Flags”, which allow the untested feature or implementation to be available only to the security team for its review.

3.2.4. Transversal Activities

The security team performs other activities which contribute to increasing the level of security awareness, meeting standards and being compliant, and, in general, increasing the security in the organization. Some of those activities are performed regularly and some others on demand but we consider them as transversal, as multiple teams and stakeholders benefit from them.

3.2.4.1. Security Training

Historically, security training sessions at Typeform were performed as a one-hour long workshop. However, last year the organization went from an on-site working approach to an almost fully remote one. This new working situation, plus the fact of not having enough security team members to handle all the workload that comes with these training sessions, made us take a different approach: asynchronous training sessions.

We split the training material and topics into small, focused, and short asynchronous training sessions. To do that we used our product: online forms.

This format, online forms, allows the engineering team members to access the training materials whenever they find them suitable. Also, with an average of 5 minutes to go through all the slides, the online forms can be consumed in a more direct and distraction-free way.

Periodically, the security team adds new training material but, once the material is prepared, there is no need to invest a regular amount of time on it: opposed to what was being done with on-site training.

3.2.4.2. Security Consultations

The security team is always available to help with any matter related to the design, implementation, deployment, testing, or any other aspect where security is considered. Same as with any doubt regarding data privacy, GDPR or any other regulation or policy. These consultations are part of the day-to-day work in the team and help promote an organization-wide security culture.

3.2.4.3. External Security Assessments

The whole Typeform application and infrastructure goes through periodic third party security assessments. Having an independent entity performing the assessments increases the level of

confidence and trust in Typeform's organization. Usually, these assessments are performed every 6 months.

3.2.4.4. Refinement of Tools and Controls

Another day-to-day task of the security team is to manage all the tools and processes offered by security. This includes the secrets detection software, the application to monitor third party dependencies, the tool to scan source code, etc. These management tasks may include adding new features, fixing possible bugs, tuning configurations or managing its users.

4. RESULTS

The four goals of the designed S-SDLC (Section 4.1) have been achieved.

The first goal fulfillment has been demonstrated by external audits. Two independents external penetration tests were performed last year, and the outcomes of both were successful: no major issues were found.

The second goal is fulfilled by leveraging Meaningful Experiments. Security is considered from a very early stage by making the security team one of its stakeholders.

As for the third goal, the S-SDLC activities have a minimal impact in the development. Meeting this objective was not an easy task. For every activity, the security team needed to find a good balance between adding security and keeping the development process agile. This balance was not found instantly, and several iterations and refinements of the activity were needed. For example, when first adding Secrets Detection to the Deploy phase many false positive detections were found. Multiple refinements of the detection rules were, and still are on some occasions, needed.

As for the fourth goal, measuring the improvement of the security team's perception throughout the organization is difficult. However, we consider that multiple factors indicate an improvement: submission of security tools improvements by the engineering team, increase in the number of security consultations, engineering teams taking ownership of certain activities like third-party dependencies analysis, etc.

5. FUTURE WORK

Typeform's S-SDLC, as any other organizations, can still be improved. For that reason, the security team has some new activities to add to the S-SDLC and improvements to be made in certain phases or activities.

Shift security even more to the left. Review which of the activities performed during the "Deploy" phase could be partially or completely moved to the "Build" phase. For example, by adding an IDE plugin that statically analyses the code as the engineers create it and raises a warning if any potential issues are detected. These types of changes need to be reviewed with the engineering team to avoid adding unnecessary friction as it might complicate the software coding process and not even translate into enough security benefits.

Study the creation of a security champion program. This program suggests the creation of a new role in the engineering teams. A security champion is someone who has enough security knowledge to help with common and basic security challenges faced during implementation,

design or any other development process. People having this role require some previous knowledge which, usually, is given by the security team. The creation of this role would be a good way of closing the gap between security and engineering team members. However, providing the necessary knowledge to these people would also require an investment of many hours by the security team. So, its viability needs to be first studied and reviewed.

6. CONCLUSIONS

This paper presents the details of a Secure Software Development Life Cycle implemented in a Software-as-a-Service organization, Typeform.

After reviewing the organization's context, four goals are set for the S-SDLC to achieve. The goals are set to be aligned with the organization's goals of a fast and frequent delivery of new software features. Then, the organization's SDLC is split in three phases: build, deploy, and run. For every phase, multiple security activities are detailed. These activities might be tasks or processes that evaluate the security of the software at a certain point of its development and produce an output.

Afterwards, how the S-SDLC goals have been achieved are detailed. We consider that our S-SDLC implementation met all the four defined goals by providing different indicators.

Finally, some improvements and future work related to the S-SDLC are discussed.

REFERENCES

[1] Martin, James (1991). *Rapid Application Development*. Macmillan. pp. 81–90.
[2] Kent Beck; James Grenning; Robert C. Martin; Mike Beedle; Jim Highsmith; Steve Mellor; Arie van Bennekum; Andrew Hunt; Ken Schwaber; Alistair Cockburn; Ron Jeffries; Jeff Sutherland; Ward Cunningham; Jon Kern; Dave Thomas; Martin Fowler; Brian Marick (2001). "Manifesto for Agile Software Development". Agile Alliance. Retrieved 14 June2010.
[3] https://www.toolsqa.com/software-testing/istqb/functional-and-non-functional-testing/. Retrieved 14 June 2021.
[4] Aaron Rinehart, Kelly Shortridge (2020). Security Chaos Engineering, Chapter 3 - SCE versus Security Theater— Getting Drama out of Security.
[5] https://www.researchgate.net/figure/IBM-System-Science-Institute-Relative-Cost-of-Fixing-Defects_fig1_255965523
[6] Donald Firesmith (23 March 2015). "Four Types of Shift Left Testing". Archived from the original on 2015-09-05. Retrieved 27 March 2015.
[7] Voitova, Anastasiia (2 November 2020). "Using SSDLC to Prepare for Security Incidents". *DZone*.
[8] Young, Scott W. H. (August 2014). "Improving Library User Experience with A/B Testing: Principles and Process". *Weave: Journal of Library User Experience*.