Deliverable 1.3

Methodology for measuring and testing tools effectiveness

DEFRAUDify - Detecting Fraudulent activities on the internet

Author: Carolyn Langen / Zahra Sedghi Gamechi (Almende)

Date: 25.03.2021

Status: Final

Version: 1.0

Table of Contents

Contents

[Table of Contents 2](#_Toc75951622)

[Acronyms 3](#_Toc75951623)

[1. Introduction 5](#_Toc75951624)

[1.1. Background 5](#_Toc75951625)

[2. Testing methodology 7](#_Toc75951626)

[2.1. Code-level evaluation 7](#_Toc75951627)

[2.2. Usability 7](#_Toc75951628)

Acronyms

|  |  |
| --- | --- |
| Acronym | Explanation |
| AI | Artificial Intelligence |
| AD |  Anomaly Detection |
| AOC | Area Under Curve |
| AML | Anti Money Laundering |
| CASE | Cyber-investigation Analysis Standard Expression |
| CTFI | Cyber Threat & Fraud Intelligence |
| CVSS | Common Vulnerability Scoring System |
| DDoS | Distributed Denial of Service |
| DW | Dark Web |
| DKIM | Domain Keys Identified Mail |
| FDS | Fraud Detection System |
| FPS | Fraud Prevention System |
| GDPR | General Data Protection Regulation |
| ICO | Initial Coin Offering |
| LEA | Law Enforcement Agency |
| ML | Machine Learning |
| MSSP | Managed Security Service Providers  |
| NLP | Natural Language Processing |
| OSINT | Open Source INTelligence |
| PCI-DSS | Payment Card Industry Data Security Standard |
| SIEM | Security Information and Event Management |
| SPF | Sender Policy Framework |
| UCO | Unified Cyber Ontology |
| VC | Virtual currency |
|  |  |

1. Introduction

The DEFRAUDify Platform will consist of several tools and solutions to detect fraudulent activities on the internet. It is essential to test the effectiveness of tools. This deliverable presents a methodology to do just that. It corresponds to activities in Task 1.3 (“Method development for measuring and testing”), which is linked to Task 1.2 (“Privacy and legal”) since tools must abide by specific legal restrictions.

This document begins with an introduction to software evaluation methods. Section 2 defines the methodology for testing tools within the DEFRAUDify project.

* 1. Background
		1. Evaluation of software for adoption by a company

Before a company adopts a given software, it is important that they consider its integration in and impact on their business. Important considerations include: (1) Whether the software is compatible with the resources in place in the client networks, such as operating system, memory and capacity; (2) The budget of clients; (3) Whether the software interacts properly with other applications, including data import/export and integrations with other tools; and (4) Whether the software increases the efficiency of operations, rather than creates additional workloads[[1]](#footnote-2).

When considering a company’s budget, the projected costs of the software development and use within the company must be considered as well, which might include project initiation costs (e.g., if the company will be replacing a legacy system), initial setup costs (including required hardware upgrades), and ongoing costs for maintenance and operation of the tool (e.g., server and network equipment maintenance fees, upgrade costs and technical support staff, administrative training)[[2]](#footnote-3). The software can also lead to cost-savings, for example by reducing employee workloads, which must also be considered along with the costs.

* + 1. Testing methodology

From an adoption perspective, software can be tested using a survey completed by relevant stakeholders2. The survey should be organized according to key areas which should be addressed. Each key area should contain relevant questions. It could also include a *software scorecard*, which includes a list of must-have and nice-to-have features sought by the company, each of which is assigned a weight according to importance of the feature. Once the software is scored, the feature scores and weights can be used to compute an overall score for the software.

The Software Sustainability Institute (SSI) proposes two software evaluation strategies: One which is criteria-based, and the other which is tutorial-based[[3]](#footnote-4). The criteria-based strategy quantitatively assesses the sustainability, maintainability, and usability of software. It focuses largely on the software at the coding level, but also contains items focused on how users interact with the software.

In contrast, the tutorial-based strategy focuses on usage of the tool from multiple perspectives, and potentially points out technical barriers to developers which may prevent software adoption. The strategy involves users completing specific tasks by following available documentation (if available) to attempt to achieve successful use of the software while evaluating the experience using a list of criteria.

The tests outlined above are typically done once software development is complete and software is ready for adoption (assuming they pass the tests). However, it is also important that software be tested during development. This is done with *white box testing*, which tests the internal structure, design and coding of the software in order to verify data flow and to improve software design, usability and security[[4]](#footnote-5). White box testing involves checking whether a set of predefined outputs produce corresponding expected outputs.

A common type of white box test is *unit testing*. Unit tests are small modules of testing code which aim to test whether the source code does what it is supposed to. They are typically automated tests which should be run each time the source code is changed to catch bugs which may have inadvertently been introduced by the new code changes. Often tests are used to verify that a given input results in a pre-specified output. Ideally, unit tests will follow best practices[[5]](#footnote-6). Some of the most important include naming tests in a self-explanatory way, focusing on a single use-case at a time, and minimizing the number of assertions in a given test.

There are other types of white box tests. Memory leak tests are conducted by quality assurance specialists who search for memory leaks, which typically result in a slow-running application. Penetration tests aim to find security threats by purposefully attacking the code and infrastructure similar to how hackers would. Mutation tests are used to determine which coding techniques are best suited to the expansion of the software.

Both the criteria- and tutorial-based strategies will be adopted by DEFRAUDify platform, with modifications where needed. The specific strategy is outlined in the following section. While white box tests are important, the appropriateness of specific types of tests is dependent on the software being developed. As such, the decision of which methods to use will be left to tool developers.

1. Testing methodology

The DEFRAUDify platform will encompass a broad range of tools, each with its own unique testing needs. These tools are, however, not yet fully defined. It is thus premature to define tool-specific SSI tests. When tools are at a greater level of maturity, tests will be defined on a tool-by-tool basis using the SSI approaches as a guide. Throughout development, tool developers should, however, continuously write and execute the appropriate white box tests in order to improve the robustness of their code. Where appropriate, the interfacing and integration between tools should also be tested. The specific tests are tool-specific, and as such will not be pre-specified in this document.

DEFRAUDify’s methodology for measuring and testing tools will involve two central aspects: code-level evaluation, and usability. These aspects will be tested with (modified) SSI criteria- and tutorial-based strategies, respectively.

All tools will need to conform to legal issues identified as part of Task 1.2. These issues include data and privacy protection, but may also include other aspects. These legal issues will be covered in a central set of testing guidelines which will apply to all tools.

* 1. Code-level evaluation

Multiple tools will be used in combination in Defraudify, therefore based on the project KPI's detailed in the full project proposal[[6]](#footnote-7) the code-level evaluation will focus on sustainability, maintainability, synergy, and interoperability. It will follow the SSI code-based approach. The template provided by the SSI will be the starting point for the definition of a tool-specific specification, where questions will be added and removed when appropriate for a given tool. A non-optional set of questions related to legal issues from Task 1.2 will additionally be included.

* 1. Usability

Tool usability will be tested according to the SSI tutorial-based approach. For each tool, this will involve a set of tasks and questions which will be completed by a typical user of the tool. This may be done in a cyclic way: test several iterations of the tool. All tools should include a set of questions which aim to test the tool’s generalizability, and where applicable, there will be a set of questions specific to whichever use cases are relevant to the given tool.

The sections and questions provided by the SSI specification will be used as a starting point, where only those items which are applicable to a given tool will be included. If necessary, additional sections and questions will be added. The resulting test specifications should, at a minimum, touch on the following criteria:

* Compatibility with existing/expected available resources, such as operating system, memory and capacity;
* Whether the tool integrates nicely with existing software, where applicable;
* Whether data import and export options are sufficient;
* Whether the tool increases the efficiency of operations;
* Whether the (suite of) tool(s) are an affordable solution, considering initiation costs, ongoing costs for maintenance and operation of the tool, equipment maintenance fees, upgrade costs and required technical support staff and training;
* The tools’ potential for delivering cost-savings.

Each of the test questions will be assigned a weight to indicate their relative importance. The final set of use-case testing questions will be each of the relevant tools separately, resulting in a separate score per tool.

1. <https://www.easytechjunkie.com/what-is-software-evaluation.htm>, accessed 25-03-2021. [↑](#footnote-ref-2)
2. <https://www.softwareadvice.com/resources/how-to-evaluate-new-software/>, accessed 25-03-2021. [↑](#footnote-ref-3)
3. <https://software.ac.uk/resources/guides-everything/software-evaluation-guide>, accessed 25-03-2021. [↑](#footnote-ref-4)
4. <https://www.guru99.com/white-box-testing.html>, accessed 07-05-2021 [↑](#footnote-ref-5)
5. [https://www.partech.nl/nl/publicaties/2020/03/10-unit-testing-best-practices#](https://www.partech.nl/nl/publicaties/2020/03/10-unit-testing-best-practices), accessed 07-05-2021 [↑](#footnote-ref-6)
6. <https://itea3.org/community/project/version/document/17470.html>, pages 43-47, 18-05-2021 [↑](#footnote-ref-7)