WP1

DIGIT B1 - EP Pilot Project 645

Deliverable 7: Comparative Study

Specific contract n°226 under Framework Contract n° DI/07172 – ABCIII

April 2016

DIGIT Fossa WP1 – Governance and Quality of Software Code – Auditing of Free and Open Source Software.

Deliverable 7: Comparative Study

Author:

Disclaimer

The information and views set out in this publication are those of the author(s) and do not necessarily reflect the official opinion of the Commission. The content, conclusions and recommendations set out in this publication are elaborated in the specific context of the EU – FOSSA project.

The Commission does not guarantee the accuracy of the data included in this study. All representations, warranties, undertakings and guarantees relating to the report are excluded, particularly concerning – but not limited to – the qualities of the assessed projects and products. Neither the Commission nor any person acting on the Commission's behalf may be held responsible for the use that may be made of the information contained herein.

Contents

List of Tables

List of Figures

Acronyms and Abbreviations

CERT	Computer Emergency Response Team
DG	Directorate General
DIGIT	Directorate-General for Informatics
EI	European Institutions
EC	European Commission
EP	European Parliament
EUPL	European Union Public License
FOSS	Free and Open Source Software
FOSSA	Free and Open Source Software Auditing
OS	Operating System
SDLC	System Development Life Cycle
WP	Work Package

1 Introduction

1.1.Objective of this Document and Intended Audience

This document represents the deliverable 7 included within Task-05 of Work Package 1 (WP1). It is the comparison of the results of task 1[1] and task 2[2] and drafting of a comparative study, which describes the comparison between the software development methodologies used in European Institutions (European Commission and European Parliament) and in FOSS communities, as well as the subsequent conclusions for the future evolution of software development methodologies at European Institutions.

It is based on the results of previous deliverables (cf. section 1.4), here presented in the form of a comprehensive and thorough study document. It also presents a set of conclusions, including best practices, lessons learnt and good methodologies to mitigate the impact of security threats.

This document is addressed to the DIGIT areas that are interested in the results of the study of the software development methodologies, related practices and tools used in the FOSS communities. It will give them enough information to improve their software development processes from a security point of view. FOSS communities can also take advantage of this deliverable by getting the recommendations obtained during the study.

1.2.Scope

The analysis covers the European Institutions' projects analysed in deliverable 3, and the FOSS communities analysed in deliverable 4. Though the terms "European Institutions' projects" and "FOSS communities" are used, they are abstractions that represent the analysed elements.

1.3.Document Structure

This document consists of the following sections:

[1] Task 1: Analysis of software development methodologies used in the European institutions

[2] Task 2: Analysis of software development methodologies used in the open source software communities

- Section 1: **Introduction**, which describes the objectives of this deliverable, intended audience and Scope.
- Section 2: **Executive Summary**, which contains the main results of the comparison.
- Section 3: **Methodological Approach to Building the Comparison**, which details how the comparison has been performed.
- Section 4: **Comparison Between European Institutions' Projects and FOSS Communities**, where their practices are compared and recommendations are provided regard to the analysis.
- Section 5: **Annexes**.

1.4.Deliverables

1. *Deliverable 3: Analysis of Software Development Methodologies Used in European Institutions*
2. *Deliverable 4: Analysis of Software Development Methodologies Used in FOSS Communities*

2 Executive Summary

This document addresses the comparison of software development methodologies between European Institutions and FOSS communities. In order to conduct this study, the information contained in the Deliverable 3 and Deliverable 4 has been considered.

The information gathered from previous deliverables is divided in ten areas: Project Management, Software Development Methodology, Software Security Definition, Software Testing, Release Management, Inspection and Code Review, Application Authentication and Authorisation, Incident Management, Problem Management, and Licensing.

For each area the comparison analyses different aspects, especially those that affect software security. The results of the comparison include a set of conclusions and recommendations, both for the European Institutions and FOSS communities so as to aid in improving their software development methodologies.

In the light of the results of this comparison, some general conclusions are provided:

- European Institutions manage Project Management activities more efficiently than FOSS communities. FOSS communities should improve this area according to their resources.
- With regard to Software Development Methodologies, European Institutions are more mature than FOSS communities. Nevertheless, some improvements are identified for European Institutions so as to develop software in a more secure way.
- As far as Security Definition is concerned, European Institutions and FOSS communities should follow the same measures to improve security. Among the analysed FOSS communities, a minority of them are highlighted for their maturity in this aspect. The security awareness in FOSS communities is higher than European Institutions.
- In the Security Testing area, European Institutions have the resources for doing any kind of software security testing, but they do not do it regularly or for all projects whereas FOSS communities should try to improve their security knowledge and testing frequency.
- European Institutions should take security into account in their Release Management practices (e.g. in their project roadmaps, security tests as release requirements) whereas FOSS communities should have more testing environments and perform more security testing.
- FOSS communities are efficient in Inspection and Code Review area, but they should improve by performing a security code review, in addition to the quality code review. European Institutions should improve this area by involving security experts in all projects to inspect the code, and promoting security awareness in all of the projects.
- A far as Application Authentication and Authorisation is concerned, European Institutions and FOSS communities should use a common module or a common mechanism for authentication, avoiding custom solutions. Also, they should use common models of authorisation, being role-based a good example. Privileges should be easy to change for any set of users.
- Regarding Incident Management, FOSS communities are better in managing incidents. European Institutions should improve their user notification process and also their incident plans with special process for critical incidents.

- In the Problem Management area, FOSS communities are more efficient at tackling problems than European Institutions, mainly because FOSS communities are used to facing problems more frequently due to their open nature.

  Nevertheless FOSS communities can improve some aspects such as following the best practices of security organisations, analyse the vulnerabilities of FOSS dependencies, and use bug-hunting programs among others. European Institutions should improve this area, by taking different actions such as having an institutional problem solving plan and taking advantage of CERT-EU support.

3 Methodological Approach to Building the Comparison

This study compares how European Institutions' projects and FOSS communities develop their software and conduct the tasks related to software security and security maintenance. The analysis will consist of a comparison of a set of areas already studied in Deliverables 3 and 4.

In order to conduct the comparison, we included a summary of the information collected for each area, highlighting some factors: Workforce, Documentation, Processes, Resources, Tools and Benefits. These factors are explained in Section 4.

Then, the analysis compares European Institutions and FOSS communities, taking those factors into account. Finally, a conclusion is obtained from the analysis and recommendations are provided according to the results.

Additionally, the analysis and conclusions is linked to the sustainability of FOSS communities. That is why some practices in FOSS communities are conducted due to the sustainability of the communities, and these practices are explained in their corresponding sections of the document, where applicable.

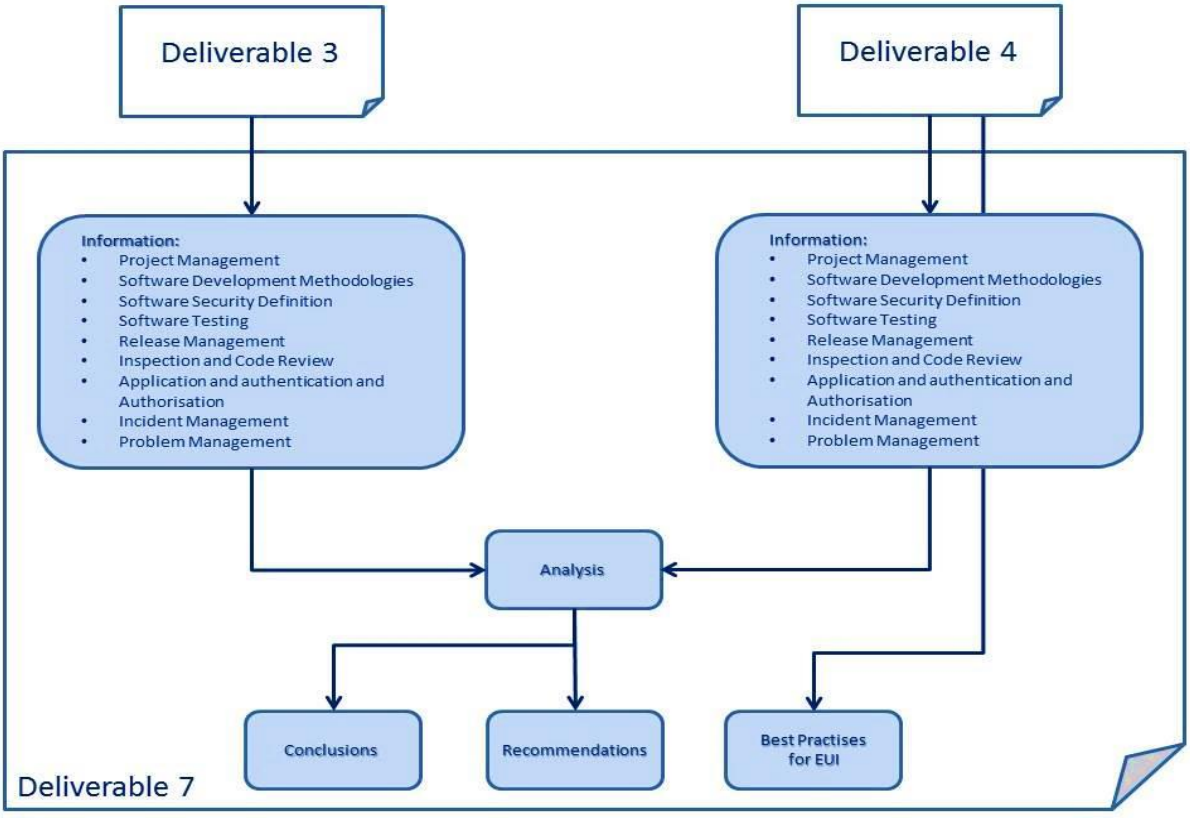

Figure 1: Methodology Diagram

4 Comparison between FOSS Communities and European Institutions' Projects

This sections aims to conduct the comparison between European Institutions and FOSS communities' projects.

Taking into account the information obtained, the areas considered in this study are:

1. Project Management: This area analyses how Project management activities are organised within European Institutions and FOSS communities.
2. Software Development Methodology: This area aims to study how European Institutions and FOSS communities develop software.
3. Software Security Definition: This area does a comparison about how security is integrated in software development.
4. Software Testing: In this area, the comparison focuses on how software is tested in European Institutions and FOSS communities.
5. Release Management: This area focuses on how software releases are managed in European Institutions and FOSS communities.
6. Inspection and Code Review: In this area, software inspection and review techniques are compared between European Institutions and FOSS communities.
7. Application Authentication and Authorisation: Some security critical aspects regarding to users, like Authentication and Authorisation, are compared in this area for European Institutions and FOSS communities.
8. Incident Management: How European Institutions and FOSS communities deal with incidents or issues is analysed in this area, considering as an incident either a bug or a vulnerability.
9. Problem Management: This area is focused on how the problems are solved, comparing it between European Institutions and FOSS communities.
10. Licensing: This area emphasizes the legal aspects of the software, comparing it for both European Institutions and FOSS projects.

For each of the areas, we analysed the following factors:

- Workforce: The human resources that are involved in the areas' procedures.
- Documentation: List of documents that support the areas' procedures.
- Processes: Main procedures of the analysed area.
- Resources: Additional resources that are used in the area (e.g. IT infrastructure).
- Tools: List of software tools that support the areas' procedures.
- Benefits: Positive effects for the analysed area.

To conduct the comparative study, we analysed the European Institutions selected during task 1[3] and the FOSS communities selected during task 2[4], which are shown on Table 1: List of European Institutions projects and Table 2: List of FOSS communities respectively.

Table 1: List of European Institutions projects

No.	Project name	Institution
1	Project 1	European Commission
2	Project 2	European Commission
3	Project 3	European Commission
4	Project 4	European Parliament
5	Project 5	European Commission
6	Project 6	European Commission
7	Project 7	European Commission
8	Project 8	European Commission
9	Project 9	European Commission
10	Project 10	European Commission
11	Project 11	European Commission
13	Project 12	European Parliament
14	Project 13	European Parliament
15	Project 14	European Commission

[3] Task 1: Analysis of software development methodologies used in the European institutions

[4] Task 2: Analysis of software development methodologies used in the open source software communities

Table 2: List of FOSS communities

Name of FOSS Community	Type
1. Apache Tomcat	One of the most popular open source Java Application Servers.
2. Bitergia	One of the most popular open source software development analytics platforms.
3. Debian	One of the most famous Linux distributions that only contains open source software.
4. Drupal	One of the most popular open source Content Management Systems (CMSs) used for websites.
5. Eclipse	One of the most popular open source IDE (Integrated Development Environment).
6. Jenkins	One of the most popular open source tools used for continuous integration.
7. LibreOffice	One of the most popular open source office suites.
8. OWASP	Open security community.
9. OwnCloud	One of the most popular open source storage cloud platforms.
10. OpenSSL	(Core Infrastructure Initiative), one of the most popular toolkits, implementing the Secure Socket Layer (SSL) and Transport Layer Security (TLS).
11. OpenStack	Open source cloud infrastructure.
12. Piwik	One of the most popular open source traffic analytics platforms.
13. Red Hat	A Linux distribution that is sold with commercial support, widely used in enterprise environments.
14. Spring	Most widely used Java framework.

In the remaining sub-sections, we analyse and draw conclusions and recommendations for each of the areas

4.1.Project Management

European Institutions

As we indicated in deliverable 3, the European Institutions' projects have standardised mechanisms to deal with project management. They use methodologies (PM² and PM for EP) based on PMBOK [1], so they follow an international professional methodology.

The roles and responsibilities are clearly defined, as well as all management processes and documentation. Dedicated expert personnel are assigned to project management activities. These are business oriented projects that require full-time working teams of different nature. These teams work in different aspects of the project such as software development and software testing among others.

As a result, European Institutions have a good maturity level regarding project management. Processes, roles and responsibilities are defined and documented, providing efficient mechanisms to manage the projects.

Highlights:

- Workforce: European Institutions rely on dedicated experts for project management.
- Documentation: Following the PMBOK approach, European Institutions they rely on PM documentation (i.e. PM templates).
- Processes: Formal processes exist and are implemented, following the PMBOK approach.
- Resources: European Institutions have all the resources needed to support project management, such as email system among others.
- Tools: JIRA.
- Benefits: Optimised project management allowing efficient project progress and control.

FOSS Communities

As far as FOSS communities are concerned, the trend is to approach project management in an informal way. They do not usually have a lot of documentation available, lacking a clear definition of roles and responsibilities. Only leader roles in the community are defined, and they are at the core of the community. This core group of contributors, who are committed with the community and trustworthy, conduct all project management-related activities. Sometimes, community sponsors participate in project management but not in all cases (especially if the community has been created by private organisations or if several organisations are actively involved in the community).

People in charge of project management have demonstrated their commitment and trustworthiness mainly through merits. This meritocratic approach is preferred because of the open and collaborative nature of FOSS communities. Most of the community workforce is made up of volunteers, and anyone can join the community. By means of this meritocratic approach, FOSS communities assign people to more critical areas of the community. In case of not carrying out this task assignment, a lack of commitment from community leaders or malicious intentions can put the community at risk.

In general, project processes (how the community works, how it is organised, etc.) are documented but not detailed, and they are not necessarily clearly defined. FOSS communities are formed by volunteers so they do not usually have full-time staff to deal with project management. Also this aspect is not interesting enough for contributors as they mostly prefer code related tasks, i.e. development, review, bug fixing. As a result, FOSS communities conduct project management using their own, informal methodology, which deals with basic aspects of project management, not tackling more mature elements.

Highlights:

- Workforce: people that conduct project management are not full time devoted or expert in project management.
- Documentation: FOSS communities produce essential PM documentation, focused on critical aspects of the community. It mainly consists of how the community works, an overview about the community, description of main community processes, etc.
- Processes: informal internal processes exist; however, they are defined only if required.
- Resources: FOSS communities' resources for project management are rare, only available if required.
- Tools: N/A.
- Benefits: informal own methodology that allows to perform project management related activities.

Analysis

In light of the results, the main difference between European Institutions and FOSS communities is the formalism of their approach. While European Institutions follow a formal and professional methodology, FOSS communities use a more basic, informal one. The European Institutions, using their methodologies, obtain more benefits in terms of project management.

This gap is the result of different factors that affect the methodology selection. The main factors are:

- Resources for project management: European Institutions have all the resources that they need for project management, and full-time experts in this field are in charge. The methodologies used institutionally provide an efficient tool to optimise project management. In contrast, FOSS communities have limited resources, and they cannot dedicate enough means to achieve the same maturity. Also they might not have full-time project management experts to control the project as it should be for formal methodologies.
- Stability of working groups: Working groups in FOSS communities are not stable, and the number of contributors can vary over time. The knowledge (know-how) in these groups can vary too, leading to unstable working groups. In this situation, roles and responsibilities have to be flexible so as to be able to adapt to the circumstances. Formal methodologies that defined, in a rigid way, roles and responsibilities might fail if working groups vary a lot. In contrast to FOSS communities, the European Institutions' projects have practically invariable resources, where the number of team members is fixed, as well as the team knowledge.
- Project Management and contributors' motivations: FOSS communities rely on volunteers, as it has been previously explained. They make this effort for free because they have some interest in the project. Their motivations are rarely aligned with project management. They prefer to focus on software development features, learning about some technology, or building a reputation in software development environments. The European institutions' projects are formed by full-time workers that are committed to the projects.

Conclusions and Recommendations

European Institutions have a mature level for Project Management. FOSS communities can use mitigation actions (provided below) to improve the way they manage the project. These actions are aimed to enhance project management and be carried out by means of promoting activities related to project management or enforcing contribution in other parts of the project, such as preparing PM templates for community's documents, planning for future tasks, controlling the execution of existing tasks, etc.

According to the analysis, FOSS communities should improve their approach to project management methodology, As a result, projects will be managed more efficiently, and the community would know what a project needs and how to optimally satisfy those needs.

FOSS communities should:

- Use a formal methodology based on PMBOK, depending on their possibilities.
- Promote project management contributions in the community.

4.2.Software Development Methodology

European Institutions

The information gathered in deliverable 3, shows that the European Institutions' projects follow formal methodologies for software development. Most of them follow the Agile principles, where Scrum is predominantly used as software development framework.

Some of the projects (about 15% of them) use the "DevOps" approach where, following Agile principles, software development teams collaborate with operation teams and quality assurance teams. This advanced approach significantly improves software quality.

As far as tools are concerned, JIRA and Confluence are commonly used to aid in software development. JIRA provides a centralised way to manage tasks and task executors, in the form of tickets. Confluence provides a knowledge base to share experience among the software development team.

With regard to human resources, European Institutions have full-time professionals that work coordinated following these formal methodologies. They have different expertise roles in the teams, where tasks are divided according to that category (Agile-based). Among these roles, there are software development leaders who organise the software development tasks.

The European Institutions' projects benefit from these software development methodologies, obtaining an efficient task management and agile approach.

Highlights:

- Workforce: dedicated full-time software development teams, and expert leaders.
- Documentation: plenty of documentation available (requirements, design, testing), according to agile principles. Also European Institutions use knowledge-based systems for sharing knowledge among contributors.
- Processes: formal processes, following agile principles and methodologies (such as SCRUM and Kanban). Some of them use the “DevOps” approach. (such as SCRUM and Kanban.
- Resources: They use a ticketing system for task management.
- Tools: JIRA, Confluence.
- Benefits: efficient software development. Knowledge is shared among the team, and tasks are managed and prioritised accordingly.

FOSS Communities

In light of the results of deliverable 4, FOSS communities use mostly informal, internal methodologies (about 80% of the FOSS communities). These methodologies are heterogeneous software development processes due to the diversity of the communities. FOSS communities are mainly formed by volunteers with different degrees of commitment and expertise, which makes it difficult to assign tasks and roles following a methodology. This diversity also affects the work approach chosen by each volunteer.

In order to coordinate the software development, they use several tools to manage software tasks. These systems play a crucial role in their methodology because they are also used as code repositories. These software tasks are usually divided in two categories: new functionalities and bug solving tasks. These tools allow defining tasks and assigning priorities, for contributors to develop all pending tasks. This tool-based approach is mostly used in FOSS communities, taking into account that these tools might provide further functionalities for other software development lifecycle phases (e.g. code review). Some of the most commonly used tools are: GitHub, Launchpad, OpenHUB, FusionForge and JIRA.

FOSS communities usually have knowledge-based systems, in the form of wikis, where some guidelines are available. Nevertheless, some of the tools previously cited, also incorporate this functionality.

Best Practice:

One FOSS community (OWASP) has recommended a good practice regarding software development methodologies which provides several benefits for software quality, such as security implementation, cost reduction and maturity of software security. This methodology is called “SecDevOps”, and it is the evolution of “DevOps” approach explained in the previous section. Aligned with Agile principles, this methodology enhances the collaboration of security teams with software development teams, operation teams and quality assurance teams.

Highlights:

- Workforce: software developers that contribute as volunteers. Most of the community leaders are volunteers strongly committed to the community, and they belong to the community core.
- Documentation: they have some documentation, mainly gathered in community wikis. The amount of documentation varies among FOSS communities.
- Processes: there are no standard software development processes in FOSS communities, leaving the approach for contributors to choose. One of the FOSS communities selected recommended a software development approach called “SecDevOps”, which is the evolution of “DevOps” that includes security.
- Resources: platform tools to manage software development, community wikis with some guidelines and the web page of the project.
- Tools: GitHub, JIRA, Launchpad, OpenHUB, FusionForge, etc.
- Benefits: This software development management approach provides a solution for coordinating heterogeneous and distributed contributions. However, these do not imply that tasks in these platforms (see Tools) have enough resources dedicated to complete them.

Analysis

In view of the results, there are some common practices that both European Institutions and FOSS communities implement. Both use different kinds of tools for the same purposes, one for managing software development tasks and the other one for sharing knowledge within the project.

However, there are notable differences as far as the formalism is concerned. While European Institutions follow formal software methodologies based on Agile principles, FOSS communities usually use an informal tool-based methodology. Furthermore, pending tasks in FOSS communities do not necessarily have contributors assigned to them. These tasks depend on the contributors’ willingness or interest in completing them.

European Institutions have enough resources to use formal methodologies for software development, achieving greater efficiency in the documentation and related processes.

The software development leaders of their projects are experts in software development management. In contrast, FOSS communities’ leaders neither are necessary experts nor are they full-time devoted, so the implementation of a formal methodology is not an easy task.

The benefits that they achieve using their methodologies are quite different. While FOSS communities cover essential necessities in terms of coordination and management, European Institutions get an optimised mechanism, allowing a faster production of software and covering all software tasks [2].

Additionally, due to the fact that not all FOSS communities use “SecDevOps”, both the European Institutions and the FOSS communities should strive to utilise this practice to ensure that security teams are involved in software development.

Conclusions and Recommendations

Both the European Institutions and FOSS communities are aligned, in terms of tool usage, with software development management. They use a centralised tool to organise software tasks (ticketing system) and another one for sharing knowledge in the form of a wiki system.

According to the previous analysis, the following actions should be carried out to improve the software development methodology in European Institutions and FOSS communities.

European Institutions should use:

- "SecDevOps", to optimise the implementation of security throughout the entire software development lifecycle.

FOSS communities should use:

- "SecDevOps" practice, according to their resources, to optimise the implementation of security throughout the entire software development lifecycle.
- Agile-based methodologies, according to their resources, so as to make software development more efficient.
- Further measures to cover pending tasks without assigned executors, such as automatic assignment of pending tasks or rewarding the execution of pending tasks among others.

4.3.Software Security Definition

European Institutions

In light of the results of deliverable 3, the definition of specific security requirements in European Institutions is properly fulfilled in more than half of analysed projects. Having specific security requirements lead to the application of security functionalities from the beginning. All project requirements are gathered by the project team, even the security ones.

This fact provides information about the maturity of security awareness in software development projects. The practice of including the security requirements within business requirements, is not a good practice. Security requirements are critical enough to be considered separately, and all of them should be satisfied during software development for all kinds of projects.

When a software project is published externally (internet), security requirements are taken into consideration more than for internal-use projects. In fact, those projects that aim to be public have to pass a risk assessment to be aware of possible issues.

The minority of the analysed projects have documentation regarding security guidelines.

Highlights:

- Workforce: project team.
- Documentation: security requirements analysis documentation, where some projects are mature in this aspect. Risk assessment is conducted in some projects.
- Processes: some of the analysed projects (about 60%) have explicit security requirements, and others perform a risk assessment.
- Resources: N/A.
- Tools: N/A.
- Benefits: those projects that have conducted a risk assessment are aware of the main security issues that they have to tackle. Those projects that have security requirements have the functionality to secure the software against potential threats.

FOSS Communities

As far as FOSS communities are concerned, more than the half of the analysed communities had defined specific security requirements. Furthermore, some of them conduct threat modelling (about 20% of them). This is a mature analysis that studies possible threats and software components, providing solutions for potential issues. Additionally, most of the analysed communities (about 85%) conduct a risk assessment to be aware of any security risks that they may have to deal with.

Security community experts and software leaders usually carry these actions out, due to the impact of security aspects in FOSS. If a FOSS is considered not secure, its usage will notably decrease and that will put the community in a bad shape. That is why security awareness is critical for FOSS communities as it can impact their sustainability.

Another factor that affects this security awareness is the community's experience. FOSS communities that have been working for several years had to deal with software vulnerabilities in the past. This enhances security awareness too. The majority of FOSS community leaders are not experts in security, they are very concerned about this topic.

This awareness can also be found in the documentation they provide for contributors. Some FOSS communities also provide guidelines about security and best practices for contributors and reviewers, to ensure a good security level in the software.

Highlights:

- Workforce: Security experts, Community leaders, Contributors.
- Documentation: Requirements analysis documentation, and risk assessment documentation.
- Processes: security requirements elicitation, risk assessment, threat modelling.
- Resources: N/A.
- Tools: N/A.

- Benefits: most of the communities are aware of the importance of security, and some of them have security requirements. Some of them have an advanced maturity level and have experience in threat modelling and the elaboration of efficient documentation about security coding.

Analysis

According to the information presented, European Institutions and FOSS communities consider security requirements in a similar approach. More than half of the projects have specific security requirements to satisfy. Some of the European Institutions' projects and FOSS communities provide guidelines for a secure coding and best practices, although these are more frequent in FOSS communities.

Nevertheless, security awareness is regarded quite differently in FOSS communities and in the European Institutions' projects. FOSS communities are more aware of the security by means of doing some actions such as: providing security guidelines to contributors, conducting risk assessment, and having standardised mechanisms to face and solve vulnerabilities. Risk assessment is conducted in most communities (about 85% of them), some of which even conduct mature analyses like Threat modelling. Furthermore, some of the FOSS communities engage security experts in the community to aid and support in software development from the beginning. Some FOSS communities involve security experts to support software development.

Conclusions and Recommendations

As explained in our analysis, FOSS communities are more aware about the importance of security than European Institutions. However, not all FOSS communities have the same maturity level. There are several factors to take into consideration in relation to these results:

- Exposition: FOSS is more exposed to attacks due to its open nature. That is due to FOSS being used in different environments and in different places.
- Experience: FOSS communities that have been working for several years already had to tackle security incidents at different points.
- Sustainability: FOSS communities depend on FOSS usage to keep working. If the software is not used, the FOSS community will lose interest and it may eventually disappear.

Thus, the following recommendations will be provided for both European Institutions and FOSS communities, though some projects/communities have already implemented them.

European Institutions and FOSS communities should:

- Consider security requirements regardless of whether the software is used internally or installed locally.

- Conduct Threat Modelling for all projects.
- Conduct risk assessment for all projects.
- Provide security guidelines to software developers during Software Development Life Cycle.
- Involve security experts from the beginning of all projects.

4.4.Software Testing

European Institutions

According to the results of deliverable 3, DIGIT test centre performs software testing to find vulnerabilities and possible bugs. This software testing process is made up of different tests:

1. **Automatic Testing** is an efficient way to conduct a set of tests without manual intervention. This saves testing resources by only checking the results. This process is carried out in most of the European Institutions' projects, and it is composed of different tests, such as regression testing, continuous integration, etc.

2. **Security Testing** refers to a series of different tests that include penetration testing, vulnerability scans and, black/white box tests. Although not all European institutions perform these security tests in their projects, more than half of the analysed projects include it.
3. **Validation Testing** aims to validate the application functionality with stakeholders. This is done before the deployment, in order to check if the application is aligned with the stakeholders' needs.

Highlights:

- Workforce: DIGIT cyber-security team, stakeholders.
- Documentation: N/A.
- Processes: testing processes are optimised mechanisms that incorporate software functionalities and the stakeholders' validation. Nevertheless, security is not always tested.
- Resources: N/A.
- Tools: JUnit, Selenium, Jenkins.
- Benefits: efficient software functionality testing, alignment with software stakeholders' needs.

FOSS Communities

FOSS communities have developed a special procedure for dealing with software testing. FOSS communities don't always have tests available for their software, so they have found other ways to test it such as automatic testing and continuous integration, code reviews and bug-hunting programs or platforms among others...

Most of the FOSS communities use automatic testing to test their software by means of regression tests and continuous integration. This approach provides a complete set of testing tools, saving resources for the communities. This is also used as a contribution acceptance test meaning that if the software that contributors provide does not pass these tests, it will be rejected.

As far as security is concerned, most of the communities review the software trying to find possible security flaws. As we noted in previous sections, security is a critical aspect of FOSS communities that impacts their sustainability. This software review is done mainly through white box testing, trying to find malicious code or "Easter Eggs".

Additionally, half of the analysed communities conduct vulnerability assessment so as to be aware of possible vulnerabilities of the software. Security specialists are required to conduct these kinds of tests, and not all communities have them among their volunteers.

For FOSS communities is hard to perform a penetration test on their software. This is due to mainly two factors:

- Software has to be running, and not all communities can do this.
- Security specialists in hacking are needed, and FOSS communities rarely have them among their volunteers.

Some communities that have enough resources try to improve this area by bug-hunting programs or platforms. In these platforms, security specialists check their software to find security issues. In turn, FOSS communities pay for each vulnerability found.

With regard to validation, they are heavily dependent on their users, who can check the development at any time, and provide opinions to developers. Validation is done implicitly; if FOSS is not aligned with users' needs, then software usage might be reduced, leaving the FOSS community in bad shape.

Highlights:

- Workforce: volunteers, security experts.
- Documentation: N/A.
- Processes: efficient test mechanism based on automatic testing, also used as an acceptance test. Security is considered during the test phase, being more important in some communities than others. User involvement is done by default, according to the communities' context.
- Resources: N/A.
- Tools: Gerrit, OpenStack Zuul, SonarQube, Jenkins, Apache Gump, etc.
- Benefits: efficient testing framework to test software, check security and engage FOSS users.

Analysis

In the light of the analysed data, European Institutions and FOSS communities perform software testing similarly in terms of automatic testing. Most of the communities and projects analysed used automatic testing to check software easily and efficiently. This process also allows them to check if new software components are compatible with the previous version.

In order to validate the software, European Institutions prepare a dedicated environment to test software involving the users. This formal approach aims to obtain an official approval from users in terms of software functionality. Nevertheless, FOSS communities use an informal approach for this validation, where it is done implicitly by FOSS users. Users can be involved in software development providing comments, testing beta versions and providing the results of the FOSS software usage.

Regarding security, European Institutions have more resources for checking security in their projects, but the awareness level is lower than FOSS communities' and not all projects conduct this kind of testing. FOSS communities try to test security according to their resources, but given their nature, it is not possible to do all the security tests required. Moreover, FOSS communities might suffer from the lack of security expertise to conduct these security tests.

Conclusions and Recommendations

As a conclusion for this section, both European Institutions and FOSS communities conduct the same kind of testing, but the approach to security is different. Automatic testing aims to save time and resources, as well as to check backward compatibility of the software. User engagement is promoted in European Institutions and FOSS communities, the process being more informal in the latter.

Security testing in European Institutions differs from that in FOSS communities in that they have more resources and expertise. However, only half of the analysed projects from European Institutions carry out this type of testing, so there is a lack of security awareness. In contrast, FOSS communities are more aware of security but they need more resources to conduct all possible security tests.

<u>European Institutions should:</u>

- Test security regularly for all projects, during the software development life cycle of the project.
- Conduct software security reviews regularly for all projects.
- Foster security awareness, by providing secure development guidelines, and analysing the results of security code reviews.

<u>FOSS communities should:</u>

- Increase security expertise in all communities, to perform security tests.
- Conduct all possible security tests for FOSS such as Black box, White box testing, etc.
- Make the test validation process more formal, using standard methodologies such as ITIL [3].

4.5.Release Management

European Institutions

European Institutions' projects use methodologies that follow Agile principles, thus reducing the Time to Market. In order to prepare a new release, European Institutions resort to several practices to manage software releases.

With regard to environments, once the software is running in the development environment it is tested in several environments with different goals. In the test environment the application has to ensure the quality of the product before the deployment in the Acceptance environment. In some projects this acts as an integration environment. Acceptance is the environment before production. Once the tests in the Test environment are successful, the version is promoted to the Acceptance environment, where business stakeholders test the functionalities according to the original requirements. This environment should be as similar as possible to the Production environment in order to mitigate the risks associated with promoting it to the Production environment.

When the acceptance/pre-production version is approved, it is promoted to this last environment in order to offer the functionalities of the application to end users. This is the environment that manages the production or "real" data of the application. For the majority of the projects the promotion to this environment is conducted by an operational team, usually an external team in charge of the system layer.

Another interesting practice that most of the projects conduct is the "continuous deployment", which is strongly related to automatic testing. In conjunction with different environments, this practice provides an efficient mechanism to test different software versions in various environments. Most of the projects use continuous deployment as a trigger to execute automatic regression, integration and unit tests.

As far as project roadmaps are concerned, most of the European Institutions' projects have a roadmap for future releases, as well as new software features. Nevertheless, barely 30% of the project roadmaps include security explicitly.

Highlights:

- Workforce: project team, operational project, stakeholders.
- Documentation: N/A.
- Processes: formal and optimised processes, where software releases are produced efficiently. Clear release goals, optimised testing process in different environments.
- Resources: different environments for software testing, servers for continuous integration.
- Tools: Jenkins, Bamboo.
- Benefits: optimisation of the way of releasing software, reducing time and resources. Clear objectives for future releases.

FOSS Communities

In the case of FOSS communities, release management is treated differently. They do not deploy their software; FOSS users do. This implies that they do not have different environments to test the software like European Institutions do. Nevertheless, they follow some practices such as “continuous integration” and roadmaps, practices also followed by the European Institutions.

Due to the nature of FOSS communities, they do not run software in test environments. They test their software using continuous integration and automatic testing. This allows FOSS communities to prepare the software for a new release and automatically check if it is ready for delivery.

Most of the analysed communities have roadmaps for future releases, and most of them also include security aspects of the software. The detail of the goals of the roadmaps varies, depending on the community mature.

Highlights:

- Workforce: FOSS users, FOSS contributors.
- Documentation: N/A.
- Processes: informal, efficiency depends on the community’s maturity level.
- Resources: continuous integration servers.
- Tools: Jenkins, Apache Gump, Gerrit, JUnit.
- Benefits for the project: preparing software for the release, and conduct automatic testing as well. Having clear release objectives, including security in most of the cases.

Analysis

In the light of the results, there are mainly three areas to compare: software environments, continuous integration/deployment and roadmaps.

As far as environments are concerned, European Institutions have plenty of environments to test software and the production environment is managed by a specialised team. FOSS communities do not have environments to test the software, so FOSS users and contributors have to conduct the tests themselves.

In regard to continuous integration/deployment, both use this practice. FOSS communities do continuous integration so as to prepare the software to be released, and they also use automatic testing. They cannot do automatic deployment for the reasons explained in the previous paragraph. In contrast, European Institutions utilise continuous integration and continuous deployment, where software moves from one environment to another after it satisfactorily passes all the tests.

Regarding to the roadmaps, most of the projects of FOSS communities and European Institutions have roadmaps to plan their future releases and objectives. Nevertheless, it is more common for FOSS communities to include security objectives in the roadmaps. The average level of awareness regarding security is higher in FOSS communities than in European Institutions, due to a higher exposure to security risks.

Conclusions and Recommendations

European Institutions have an efficient system to test software, which includes several environments with different goals. This system also uses continuous integration and continuous deployment in the testing process, preparing the software for being released, and easing the progression of the software from one environment to another. They can test different versions at the same time, reducing the Time to market.

Most FOSS communities do not have the infrastructure or the resources necessary to do this kind of testing. Nevertheless, most of them use continuous integration so as to do the automatic testing and to prepare the software for release.

Roadmaps for future releases and objectives are used in most of the European Institutions and FOSS community's projects, but FOSS communities include security objectives more frequently. The projects of the European Institutions take security into account especially when the software is going to be exposed to the internet or it is critical software for them.

In view of the results, the following recommendations should be considered by European Institutions and FOSS communities:

European Institutions should:

- Include security in project roadmaps.
- Include security tests as release requirements for all projects.

FOSS communities should:

- Have test environments, if possible
- Implement security testing.

4.6.Inspection and Code Review

European Institutions

Taking into account the study elaborated in deliverable 3, this section provides a summary of the results of the assessment of software inspection methodologies in European Institutions. European Institutions' projects are heterogeneous in this regard, lacking a common process for inspection purposes.

In view of the data gathered in deliverable 3, code review is done during the development phase of almost half of the analysed projects. Almost 30% of the analysed projects have a different team to inspect the code, 20% of them do not review the code, an external team inspects the code in two of them, and in one of the analysed projects the code is inspected by a FOSS community.

In regard to resorting to security experts for code review, these projects show heterogeneous results. Almost 60% are reviewed by DIGIT, 30% are reviewed externally, 20% are not reviewed by security experts, and two of the projects do it internally.

The point in time selected for these experts to review the project varies as well: 60% of the projects analysed left the review for the end. Only two of them integrated an expert security review throughout the entirety of the project. Another two projects integrate expert security reviews in the initial phases (analysis and design), and in one of them security experts review the project during the development as well.

Considering the previous information, it should be noted that code review and security review are not institutional processes, leaving it up to the project manager to decide how to proceed. In most of the cases, there are no formal processes. The lack of security awareness and institutional processes entails that the project manager has to choose how to properly implement code and security inspection in the project. Thus, these processes depend on the knowledge and experience of the project manager and its team.

Highlights:

- Workforce: mainly project teams and DGIT.
- Documentation: N/A.
- Processes: informal and non-institutional, depending on the project manager and project team knowledge and experience.
- Resources: DIGIT systems
- Tools: Crucible, SonarQube, GitHub, SVN.
- Benefits: dependent on the mechanisms implemented, code quality and security control.

FOSS Communities

For FOSS communities, this is a critical aspect for their projects. The sustainability of the community depends to a certain extent on FOSS quality, and how new contributions are integrated. . The efficiency of the “open” nature of the community when it comes to receiving useful contributions depends on code review and inspection.

All code contributions have to be reviewed by several community members before being accepted. This is a formal process for some FOSS communities, where they have documentation and roles established to conduct the necessary activities. In most of the FOSS communities, community members review the code and, in some of them, a specific team checks the code as well.

In 75% of the analysed communities, security experts review the project. They mainly do this review during the development or at the end of the development phase.

Additionally, due to the open nature of FOSS communities, contributions are public. For software contributors, this implies that the quality of the code that they provide will be tested by anyone within the community. Their personal reputation can be damaged if the quality is poor.

Highlights:

- Workforce: specific team, community members, some security experts.
- Documentation: N/A.
- Processes: mature process, in some cases it is a formal process.
- Resources: open source repositories.
- Tools: SonarQube, GitHub, Launchpad, OpenHUB.
- Benefits: standardised process to check each code contribution during the development, so as to ensure software quality and security.

Analysis

In the light of the previous information, two different factors should be considered: how the code review is done, and how security experts review the software. Code review is, in general, a more mature process in FOSS communities than in European Institutions. As we previously explained, code review is a test that a contribution has to pass in order to be accepted.

European Institutions rely on the project team's experience and knowledge to ensure the efficiency of the mechanism selected for code review, while FOSS communities have, in general, years of experience doing it on their own. Moreover, the personal reputation of the contributor is affected by the quality of the contribution.

Expert security reviews are more frequent in FOSS communities than in European Institutions, but FOSS communities should review the security of the code for each specific contribution. This is difficult for them, given the resources required. European Institutions have enough resources to conduct security reviews; however they lack security awareness.

Conclusions and Recommendations

In conclusion, we believe that, in order to improve both review processes, European Institutions should focus on how the code is reviewed, emphasising the importance of security. They have the necessary resources, but they suffer from a lack of awareness regarding security.

In contrast, FOSS communities are more aware of the importance of security, but they should go one step further in terms of security reviews. One efficient way could be to implement an explicit security review of each contribution and to explicitly incorporate security in the review process.

To sum up, the maturity level of software inspection and code review is higher in FOSS communities than in European Institutions. This is why FOSS communities consider it essential to ensure software quality and security, since it has a direct impact on the sustainability of the communities.

European Institutions should:

- Make the code review process more formal, for all projects.
- Involve security experts in all projects to inspect the code.
- Promote security awareness in all of the projects.

FOSS communities should:

- Conduct a specific security review for each contribution, if possible.

4.7.Application Authentication and Authorisation

European Institutions

This section gathers information about how European Institutions' projects deal with two important security aspects: authentication and authorisation. Authentication is the process to verify the user identity, and authorisation is the process that allows the access of application resources by checking the user privileges.

In regard to authentication, most of these projects rely on a common authentication module, ECAS. This module uses CAS, a well-known protocol, to perform authentication. This provides a tested mechanism to enable access to users, avoiding common authentication errors.

As for authorisation methods, in most of the analysed projects authorisation is based on roles. This mechanism is a widely used approach where roles are related to a set of privileges. Users have roles according to their use cases and necessities, and thus a set of resources are granted by default. This mechanism is an efficient manner to control access resources and it is also easy to maintain.

In this sense, European Institutions rely on widely-used practices and modules to conduct sensitive operations, like authentication and authorisation.

Highlights:

- Workforce: N/A.
- Documentation: N/A.
- Processes: mature process, in some cases it is a formal process.
- Resources: common modules, protocols and practices to perform authentication and authorisation.
- Tools: N/A.
- Benefits: well tested mechanisms for authentication and authorisation processes, avoiding common errors.

FOSS Communities

With regard to FOSS communities, the usage of authentication modules varies among communities. About 40% of the analysed communities use an authentication module developed by them; another 40% use an external module, and the remaining 20% of them use a module of the same community, but from a different project.

Regarding the authorisation approach, role-based is preferred to others. User groups are only used in about 20% of analysed communities. In FOSS communities the role-based authorisation approach is the trend, using this structured way to assign privileges.

According to the data, several FOSS communities use external modules for authentication purposes, instead of developing their own ones. Furthermore, role-based authorisation is the most widely used authorisation mechanism among the analysed FOSS communities.

Highlights:

- Workforce: N/A.
- Documentation: N/A.
- Processes: mature process, in some cases it is a formal process.
- Resources: common authentication modules for several communities, and different authorisation models (role-based and group-based).
- Tools: N/A.
- Benefits: several communities use common modules for authentication processes. Thus, they use a tested authentication module. Role-based access control approach followed by several communities provides a standard mechanism to manage privileges and users in an efficient way.

Analysis

European Institutions and FOSS Communities are aligned in terms of authentication and authorisation. The trend for authentication is to use an external, well tested module in both cases, although it is more frequent in European Institutions. This avoids common authentication errors in the implementation.

Similarly, the trend for authorisation purposes, both in European Institutions and FOSS communities, is to use the role-based model, although it is more frequent in European Institutions. This provides an optimised model to match users and privileges, according to their needs. Besides, privileges are managed in a centralised way that allows modifying the access to resources by modifying the roles.

Nevertheless, some FOSS communities use other approaches. This can be explained due to the fact that the analysed communities are diverse, and thus different kinds of software are analysed too. In contrast, most of the European Institutions' projects use a similar kind of software (web applications).

Conclusions and Recommendations

In view of the previous analysis, European Institutions are aligned in terms of authentication and authorisation aspects. In European Institutions the trend regarding authentication is stronger than in FOSS communities. Thus, some recommendations are provided for those projects/communities that do not follow the trend.

FOSS communities and European Institutions should:

- Use a common module or a common mechanism for authentication, avoiding custom solutions, if applicable.
- If possible, use common models of authorisation, being role-based a good example. Privileges should be easy to change for any set of users.

4.8.Incident Management

Based on ITIL [3], an incident is a single occurrence of a difficulty, which is affecting the normal or expected service of the user. The usual priority when an incident occurs must be to restore normal service as quickly as possible, with minimum disruption to the users.

European Institutions

European Institutions' projects have different ways for dealing with incidents such as a bug or vulnerability. This process can entail several levels of support for some projects, but more than half of the analysed projects have only one. About 35% of them have two levels, and one of them has three levels. Analysing the entry points of bug and vulnerability reporting, most of the projects use the helpdesk as an entry point (about 80%), while 40% of them use JIRA, and only one of the analysed projects uses a mailbox for this purpose.

In order to deal with major incidents, most of the analysed projects raise the priority of the tasks that need to be solved. About 60% of them speed up the task solving process, skipping some environments, and about 30% shut down the system for major incidents.

It seems obvious that when it comes to facing common incidents there is a standardised procedure that can be complex, with different support levels for some projects (about 40%). Formal entry points are defined, where tasks are organised and prioritised. Major incidents are often speeded up by raising the priority and skipping environments, as more than half of the projects do.

Nevertheless, almost one-third of the analysed projects interrupt the service for major incidents. This is a bad practice, since interrupting the service for a major incident proves the lack of a mature response plan. The maturity of an incident response plan is strongly related to security awareness, and in this aspect some European Institutions' projects are not as mature as FOSS communities.

As far as user notification is concerned, the analysed projects follow different approaches. About 30% of them communicate the incidents to end users, another 30% inform the person responsible for the

application on the business side. Providing information about incidents is a good practice that improves reliability and trustworthiness.

However, two of the analysed projects solve incidents without any notification. This is a bad practice, because the staff from the business side should be informed in some way, to be aware of possible risks.

European Institutions' projects have a plethora of possible channels to notify the staff in the business end. The decision is taken by project leaders, and thus there is a wide variety among projects. The most widely used channel is email, used in 50% of the projects. Yammer, an enterprise social network from the USA, is also used to notify incidents in 40% of them. 40% of the analysed projects also use a static webpage to inform stakeholders from the business side. Intranet is used by 30% of the projects, and two of them use Helpdesk to communicate incidents. Finally, one of them uses Confluence and another one uses IRM.

Highlights:

- Workforce: helpdesk, project team.
- Documentation: documentation about procedures for support activities, incident response plans.
- Processes: No institutional processes. There are standardised processes where maturity varies depending on the project. Major incidents are managed faster than common ones, but this also varies among projects.
- Benefits: support activities implemented to face incidents where critical tasks are speeded up. The efficiency in this aspect varies for each project.
- Resources: email, intranet.
- Tools: Yammer, Confluence, IRM.

FOSS Communities

FOSS communities are used to dealing with incident management, and bugs and vulnerabilities are reported to the community by FOSS users. A good example is the different approaches used by communities, where most of them (about 80%) use bug management platforms. In these platforms, incidents (bugs and minor vulnerabilities) are reported by users, and the FOSS community organises and prioritises them. Several communities (about 65%) use emails as an entry point, especially for major vulnerabilities.

Due to the open nature of the FOSS communities, major vulnerabilities have to be managed carefully. This is why the existence of a critical vulnerability has to be communicated to the security team of the community via email. This prevents the public disclosure of information regarding the vulnerability before it is solved.

In regard to user notification, FOSS communities mainly use the community website to inform about incidents (about 80%). Release notes are used by some of the communities analysed (about 40%), as well as email lists (about 30%). JIRA is rarely used in these communities (only in two).

FOSS communities do not have support levels as European Institutions do; instead they have forums where users can post messages for the community, and a wiki where users can check information about FOSS.

Highlights:

- Workforce: FOSS community.
- Documentation: information documented in the webpage, wikis and the communities' forums. This information consists of bug notification for users, and bug solving and incident response plans for contributors.
- Processes: standardised processes for facing bugs and vulnerabilities.
- Benefits: clear entry points for bugs and vulnerabilities, which are managed and prioritised in a central manner. Major issues are managed privately until their resolution.
- Resources: email system, wiki systems, and forum sites.
- Tools: Bugzilla, GitHub, FusionForge, Launchpad, OpenHUB.

Analysis

European Institutions and FOSS communities have different processes when it comes to dealing with incidents. While European Institutions mainly use helpdesk as the entry point for incidents, FOSS communities do not have enough resources to have helpdesk, thus they use bug management platforms as an alternative. Nevertheless, FOSS communities usually have a special entry point for major issues, handled by a specialised team. That is why vulnerabilities might leak before being solved.

FOSS communities do not run their software in their production environments, so they can solve issues as soon as possible. In contrast, European Institutions run their software in the production environment, thus they have to deal with issues while software is in production. The mechanism to solve issues is contained in the various incident response plans. About 30% of the European Institutions' projects shut down the system for major incidents.

European Institutions use helpdesk to provide support to users, and the support levels vary depending on the project. In contrast, FOSS communities mainly use forums and wikis to provide support to users.

While FOSS communities always inform users about issues, not all European Institutions' projects notify the business side staff about them. Two of the analysed projects do not communicate any information in relation to the issues. The channels used for notifications are similar in both European Institutions and FOSS communities. They have some common channels, such as email lists and web-based approaches (wikis, static pages, and intranet).

It is important to note that 40% of the analysed projects from European Institutions use an enterprise social network from the USA (Yammer) to inform users of issues (bugs or vulnerabilities) in the

software. So the European Institutions software vulnerabilities are in a server outside the EU, and therefore subject to sensitive data leakage.

Conclusions and Recommendations

In view of the previous analysis, FOSS communities tend to manage incidents better than European Institutions. FOSS sustainability depends on FOSS reliability and trustworthiness, and thus incident management is critical for FOSS communities. However, FOSS communities are more exposed to incidents than European Institutions.

The following recommendations are provided for the European Institutions in order to improve their incident management processes.

European Institutions should:

- Find a substitute tool for notifying issues (bugs or vulnerabilities) in the European Institutions' software.
- Notify end users about issues (bugs or vulnerabilities), for all projects.
- Prepare adequate incident response plans for all projects, trying to avoid shutting down running software.
- Have special, standardised processes for major incidents. It is advisable to have common formal institutional processes so as to face major issues effectively, managed by a special team.

4.9.Problem Management

Based on ITIL [3], a problem is the underlying cause of one or more incidents, the exact nature of which has not yet been diagnosed. Restoring normal service to the users should normally take priority over investigating and diagnosing problems, although this may not always be possible.

European Institutions

European Institutions use different mechanisms to detect bugs or security flaws. Most of the European Institutions' projects (about 90%) detect problems by means of regression tests. These regression tests are mainly done using automatic testing and continuous integration systems.

Other channels are also used, such as security information updated from providers and external sources. About 35% of the projects retrieve updated information from their providers (Drupal, EM, Alfresco, etc.), and also about 35% of them get information from external sources. These external sources are usually vulnerability repositories and CERTs (computer emergency response teams), like CERT-EU [4].

Once the problem is found, the problem solving plan is the procedure to solve it. Depending on the severity of the problem, there are two possible approaches: standard release or a special process. Most of the analysed projects (about 80%) prefer standard releases, and about 35% of the projects have a special process to face major problems.

Highlights:

- Workforce: project team, some projects involve their providers.
- Documentation: problem solving plans.
- Processes: standard Release for minor problems, and a special process for major ones.
- Resources: OWASP [5] recommendations, DIGIT recommendations, Provider Recommendations (Drupal, Alfresco, Piwik).
- Tools: Selenium, JUnit.
- Benefits: standardised project mechanisms to identify problems, categorise them and solve them.

FOSS Communities

FOSS communities have several ways to detect bugs or vulnerabilities, but one of the most important sources is FOSS users. Most of the analysed FOSS communities (about 95%) detect flaws through their users, who usually run FOSS in their environments. This plethora of different environments allows the community to test the software in different contexts. Thus, since it is a FOSS solution, it is more secure thanks to the larger number of users running and "testing" it.

They also take advantage of their contribution system, where a contribution has to be examined before being accepted. This mechanism allows detecting possible problems before accepting the contribution. Most of the communities detect potential problems using this method (about 80%).

Regression tests are used in FOSS communities, and most of them (about 85%) detect potential flaws thanks to this practice. Once a contribution has been approved, it is added to the integration system and to the automatic testing.

In some of the analysed communities, external security experts detect problems. Only 22% of the communities find potential problems by this means. Sometimes this is conducted by bug hunting platforms (like HackerOne [6]), where problems detected have rewards.

The problem solving plan is flexible, and remediation is delivered depending on the severity. Solutions for minor problems are queued for the following release, and severe ones are delivered as soon as possible. The remediation can be delivered either as a new release or as a patch, the first approach being the most frequent.

The process to solve critical problems is usually a special process managed by a specific team. Users communicate the problem via a dedicated channel (mainly email) and the team solves the problem as soon as possible. This team is made up of trustworthy community members that work together on the solution. Once the solution is released, the problem is made public.

Highlights:

- Workforce: FOSS users, community contributors.
- Documentation: problem solving plans.
- Processes: standard Release for minor problems, and a special process for major ones.
- Resources: continuous integration systems, bug tracker platforms.
- Tools: Bugzilla, Launchpad, GitHub, OpenHUB.
- Benefits: standardised mechanisms of their own to identify, categorise and resolve problems.

Analysis

In the light of the information gathered, there are mainly two aspects to compare: how potential problems are detected and how they are solved. With regard to how the problems are detected, European Institutions and FOSS communities use a similar approach, using regression tests. Most of the analysed FOSS communities and European Institutions' projects detect potential flaws in this way.

FOSS communities have a special process to deal with major incidents. However, not all the European Institutions' projects have this process.

In contrast, they differ in several aspects: some European Institutions projects rely on their providers to detect problems, and they check external sources of information to gather information on them. FOSS communities do not usually have providers to help them deal with problems or to provide information about them. Some FOSS communities inform and check external sources with regard to problems, but this depends on the community.

Additionally, FOSS communities take advantage of their open nature, where software contributions are analysed before being accepted, and FOSS software is run (and tested) in several different production environments (FOSS users' environments). Thus, potential flaws can be easily detected. The usage of bug hunting platforms to reward detected problems is a good practice, since it fosters the contributions aimed at securing the software.

Conclusions and Recommendations

In view of the previous analysis, FOSS communities and European Institutions are aligned as far as regressions tests are concerned, and also in terms of special processes for critical problems. However, not all the European Institutions' projects have a special process for critical problems. This absence of special processes might be caused by a lack of awareness regarding the security risks that they have to face.

They follow a similar practice in terms of using problem solving plans to indicate how to manage a problem once it arises. These are specific plans for projects and FOSS communities. European Institutions should consider providing an institutional problem solving plan for all of their projects so as

to be efficient. FOSS communities should standardise their plans and try to follow a formal standard (like ITIL [3]).

European Institutions take advantage of their nature, involving providers (if any) and CERTs (like CERT-EU [4]) so as to be more aware of their software's weaknesses. However, this is not yet included in all projects. All of them should involve their providers in problem detection and remediation processes, and stay up-to-date in relation to problems by checking the information of CERTs.

FOSS communities also take advantage of their nature, where potential flaws can be detected during code review, and in the users' production environments. The security of the FOSS software depends on several factors, such as how widely FOSS is used and how big the FOSS community is. Moreover, some FOSS communities use bug hunting platforms to increase the security, and this practice should be used in all communities (if possible) and all European Institutions' projects.

According to the results, several recommendations can be provided for European Institutions and FOSS communities:

<u>European Institutions should:</u>

- Have a special process for solving critical problems, for all projects.
- Always follow organisations that provide best practices with regard to secure coding, like OWASP, for all projects.
- Consider providing an institutional problem solving plan for all of their projects so as to be efficient.
- Take advantage of their resources and involve their providers in problem detection and problem remediation processes. Also check CERT´s' information, especially CERT-EU. This should be done for all of the projects
- Always check vulnerability repositories of software used in the projects and its dependencies, for all projects.
- Consider the usage of bug hunting platforms to find software problems.

<u>FOSS communities should:</u>

- Always inform vulnerability repositories of FOSS problems, for all communities.
- Always follow organisations that provide best practices about secure coding, like OWASP. For all FOSS communities.
- Always check vulnerability repositories of software used in the projects and its dependencies, for all communities.
- If possible, make use of bug hunting platforms to detect problems in the software.

4.10.Licensing

European Institutions

Due to the nature of the European Institutions' projects, software is proprietary by default. Nevertheless, several of the analysed projects (about 46%) collaborate with FOSS communities. This collaboration is done in the form of code sharing and information sharing.

With regard to code sharing, European Institutions sometimes share new functionalities or updates for the FOSS software that they use. Software developers provide personal contributions, since they cannot operate on behalf of the European Institutions because of legal constraints. As far as information sharing is concerned, some European Institutions' projects inform FOSS communities of procedures, methodologies and architectures.

Joinup [7] is a collaborative platform created by the European Commission and funded by the European Union via the Interoperability Solutions for European Public Administrations (ISA) Programme. It offers several services that aim to help e-Government professionals to share their experience with each other. This platform is used by some projects (about 30%) to share information.

It is important to highlight that two of the projects share the entire application licensed as European Union Public License (EUPL-1.1), and in this case the contribution is made on behalf of the institution.

The features of this license can be reviewed in Annex 3: Open Source Licenses.

FOSS Communities

FOSS communities have a plethora of different open licenses and different versions. These licenses describe how the FOSS can be used. In this study, we analysed the most popular ones, as follows:

- Apache License 2.0
- BSD 2 – Clause (FreeBSD/Simplified)
- BSD 3 – Clause (revised)
- Eclipse Public License (EPL - 1.0)
- GNU General Public License (GPL v2)
- GNU General Public License (GPL v3)
- GNU Lesser General Public License v2.1 (LGPL – 2.1)
- GNU Lesser General Public License v3 (LGPL – 3.0)
- MIT license (MIT)
- Mozilla Public License 2.0 (MPL - 2.0)

The features of the analysed licenses can be reviewed in Annex 3: Open Source Licenses.

Conclusions and Recommendations:

Due to the nature of FOSS communities and European Institutions, recommendations on which is((are) the best license(s) to use is not possible, the reason being that software licensing is based on the strategy or objectives of the project or community.

However, some general recommendations are provided.

European Institutions should:

- Consider what part of their software can be shared (i.e. correctly licensed) so as to join efforts in common necessities, among projects and the European society.
- Before using a FOSS, review the respective license and comply with it.
- Analyse the consequences of using software under some Open Source licenses, as there might be obligations resulting from its usage, e.g. the new software and all related components has to be shared and this might impact the European Institutions Intellectual Property Rights.

FOSS communities should:

- Use a common license according to community strategic objectives. Review that contributions are license-compatible.
- Engage the advice of some type of legal support group within or outside of the community
- Dedicate people to watch for the license adherence.

5 References

[1] PMI, A Guide to the Project Management Body of Knowledge (PMBOK® Guide)—Fifth Edition (ENGLISH), Project Management Institute, 2013.

[2] B. Lucas, “Agile Savings versus Traditional Methods,” [Online]. [Accessed 15http://www.compaid.com/caiinternet/ezine/AgileSavings-Lucas.pdf April 2016].

[3] ITIL, “Itil Books,” [Online]. Available: http://www.itil.org.uk/.

[4] European Comission, “CERT-EU,” European Comission, [Online]. Available: https://cert.europa.eu. [Accessed 15 April 2016].

[5] OWASP, “OWASP,” Free and open software security community, [Online]. Available: https://www.owasp.org.

[6] HackerOne, “HackerOne,” HackerOne, [Online]. Available: https://hackerone.com/. [Accessed 15 April 2016].

[7] E. Comission, “joinup,” [Online]. Available: https://joinup.ec.europa.eu/.

[8] IBM, “IBM Rational,” [Online]. Available: http://www.ibm.com/software/rational.

[9] European Commission, “joinup,” [Online]. Available: https://joinup.ec.europa.eu/.

6 Annexes

6.1. Annex 1: Summary of Recommendations

6.2. Annex 2: Best Practices Used by FOSS Communities that Can Be Useful for European Institutions.

6.3.Annex 3: Open Source Licenses.

Table 3: Open Source Licenses

Features/Licenses	EUPL 1.1	Apache License 2.0	BSD 2	BSD 3	EPL 1.0	GPL v2	GPL v3	LGPL 2.1	LGPL 3.0	MIT	MPL 2.0
Commercial Use	√	√	√	√	√	√	√	√	√	√	√
Modify		√	√	√	√	√	√	√	√	√	√
Sublicense (include in a more restrictive license)		√			√	X	X	X	X	√	√
Distribute original or modified works	√	√	√	√	√	√	√	√	√	√	√
Use patent claims (the rights to practise patent claims of the contributors)		√			√		√		√		√
Hold liable (software/license owner cannot be charged for damages)	X	X	X	X	X	X	X	X	X	X	X
Use trademark		X		X	X						X
Place warranty (the ability to place warranty on the software licensed)				√		√	√		√		√
Private Use	√	√			√					√	
Mandatory											
Include License	√	√	√	√	√	√	√	√	√	√	√
Include Copyright	√	√	√	√	√	√	√	√	√	√	√
State changes	√	√				√	√	√	√		
Include Notice File (”NOTICE”) with attribution notes		√						√			
Include Install Instructions					√		√		√		
Include original (copies of the original software)					√	√	√	√	√		√
Compensate for damages (If developers include the software in a commercial product, developers must defend and compensate the EPL contributor from lawsuits/damages caused by your commercial offering					√						
Disclose Source	√				√	√	√		√		√