WP1

DIGIT B1 - EP Pilot Project 645

Deliverable 4: **Analysis of Software Development Methodologies Used in the FOSS Communities**

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

February 2016

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

Deliverable 4: Analysis of Software Development Methodologies Used in the FOSS Communities

Author:

Disclaimer

Contents

List of Tables

List of Figures

Acronyms and Abbreviations

API	Application Programming Interface
EUI	European Institutions
EP	European Parliament
DG	Directorate General
DAC	Discretionary Access Control
ESAPI	Enterprise Security Application Programming Interface
FOSS	Free and Open Source Software
FOSSA	Free and Open Source Software Auditing
MAC	Mandatory Access Control
OS	Operating System
RBAC	Role-based access control
SDLC	System Development Life Cycle
SEO	Search Engine Optimization
WP	Work Package

1 Introduction

1.1. Objective of this Document and Intended Audience

This document represents the deliverable 4 included within TASK-02: Analysis of software development methodologies used in the Free and Open Source Software – FOSS communities.

The objective of this document is to analyse the software development methodologies, tools and best practices used in the FOSS communities that were selected and prioritised in Deliverable 2.

This document is addressed to the DIGIT and ITEC departments that are interested in reviewing and analysing the results of the study of the software development methodologies, related practices and tools used in the FOSS communities, which, together with the results of Deliverable 3, will give them enough background information to understand and review Deliverable 7.

1.2. Scope

The analysis covers the FOSS communities that were selected during the development of Deliverable 2. To accomplish this analysis, a representative of the community was interviewed.

Throughout the document, the term “FOSS communities” refers to the FOSS projects, communities and foundations that fall within the defined scope. Red Hat, a private OSS organisation, was included in the analysis at the request of DIGIT.

1.3. Document Structure

This document consists of the following sections:

- Section 1: **Introduction**, which describes the objectives of this deliverable, intended audience and Scope.
- Section 2: **Methodological Approach to Building the Analysis**, which describes the steps that we followed to conduct the analysis of the different methodologies, tools and best practices used by the selected FOSS communities, according to the scope.
- Section 3: **Software Development Methodologies, Best Practices and Tools** used in the FOSS communities.
- Section 4: **Analysis** of the identified software development methodologies used in FOSS communities.
- Section 5: **Bibliographical** references.
- Section 6: **Annexes**.

1.4. Key Success Factors

All steps described in Section 2 - Methodological approach to building the analysis, will ensure the fulfilment of key success factors related to this deliverable:

- Having a complete stock of methodologies used both in the European Institutions and FOSS communities that were selected for this project.
- Including a variety of best practises typologies: technical, organisational and about the governance and quality of free and open source software (e.g.: synchronisation with private organisations; guidelines for secure software development; secure integration and interoperability of different components; sustainable ways of FOSS governance and professional services).
- Integrating practical best practices within existing processes, procedures and tools (e.g.: CEV database).

1.5.Deliverables

1 *Deliverable 2: Approach Towards the Execution of Task 2*
2 *Deliverable 3: Analysis of Software Development Methodologies Used in the European Institutions*
3 *Deliverable 7: Comparative study*

2 Methodological Approach to Building the Analysis

The goal of this document is to analyse all information gathered during the interviews and research conducted by everis' teams that relate to this study. This analysis will provide valuable information from the identified FOSS communities in regard to:

- Software development methodologies in use
- Best practices in use
- Tools in use
- Release management
- Incident management
- Security aspects related to software development
- Their points of view on how European Institutions can contribute to ensure that critical software can be trusted.

2.1. Selection of Projects, Engagement with Free and Open Source Software Communities and Information Gathering

For this step, the following activities were conducted:

- Deliverable 2 provided a list of 14 FOSS communities and organisations to be analysed.
- In order to engage the communities´ representatives, everis sent an executive summary explaining the importance of the FOSSA project, and requesting their availability for an interview to gather information on their particular project, community or organisation.
- During the interview rounds, 14 out of 15 projects were covered.
- The everis team of FOSS experts provided information on best practices, methodologies and tools used by some of the communities.
- The everis project team researched the communities that were not interviewed, to gather information on their best practices, methodologies and tools.

2.2. Information Classification and Filtering Process

The following figure shows which information sources were used to conduct the analysis.

Figure 1: Methodological approach used to build the analysis - Information sources

- **Interview Results**: During the interviews with the representatives of the FOSS communities, we used a questionnaire to obtain the relevant information for the study. Since the interviews were conducted as an open discussion, the information gathered was filtered and classified to conduct the analysis. For this purpose, a spreadsheet was created to count the number of projects using a specific methodology, practice or tool under analysis. Common criteria were taken into account, but the particularities of each community were also included, as they could add value to the study. After filtering and classifying the data, each methodology, practice or tool used by a community was compared with the ones used by other communities; this allowed the calculation of the percentage of usage within the communities analysed. This percentage is an indication of how often the analysed variable is used or followed by the projects selected among the FOSS communities.

- **Documentation Analysis**: In order to complete the information related to the identified methodologies, best practices and tools, public documentation found on the communities' websites was analysed to fulfill the aspects mentioned above. everis's team of experts also provided information for the analysis.

2.3. Analysis of the Information

Sections 3 and 4 of this document are structured following two main objectives:

- **Software development methodologies, best practices and tools used in the FOSS communities**: For each of the methodologies, best practices and tools gathered from the interviews, a form is created in order to complete the information about each variable.
- **Analysis of identified software development methodologies, best practices and tools used in the FOSS communities**: This section is structured according to four main points in conducting the analysis:

    - **Project Management:** Analyses the methodologies used within the different phases of project management.

- **Software Development Lifecycle**: Analyses the methodologies, practices and tools used within the different phases of the project development.
- **Project Maintenance**: Analyses the methodologies, practices and tools used to ensure the sustainability of the projects and their quality.
- **How European Institutions contribute to FOSS Communities**: Analyses the actual contribution of the projects and teams to FOSS communities.
- **Relevant opinions and advices from interviewees**: Contains interviewees' personal opinions and advice expressed during the interviews.

The usage of each analysed variable is represented by a numeric value and a percentage. To represent these numbers, we used three different approaches:

- **Tables**: Representing the percentage of usage for the total number of projects analysed. Note that the variables are not mutually exclusive; therefore, a project can use one or more of them.
  To calculate this percentage, we used the following formula:

%usage = nCoincidences * 100 / nProjectsAnalysed

- **Pie Charts**: The percentages of usage are represented graphically, allowing clear and concise view of the results. The variables analysed using this approach are exclusive; therefore, a project can only use one of them.

3 Software Development Methodologies, Best Practices, Frameworks, Libraries and Tools Used in the Projects Analysed from the FOSS Communities

Information about software development methodologies, best practices, libraries and tools used in the FOSS projects analysed, is gathered in this section. This information comes from different interviews with the FOSS communities and documentation analyses found on their websites. The criteria to fulfill the templates is the following:

- If the FOSS community uses the feature, it is marked with “X”
- If the FOSS community has been interviewed and the answer was not conclusive, it is marked with “?”
- If the FOSS community has not been interviewed and/or the information was not found, it is marked with “N/A”

3.1. Methodologies Used by the Analysed FOSS Communities During the Software Development Lifecycle

M1. Methodology Name:			**Scrum**			
Use		**Objectives**		**Benefits**		
Software development		Manage the software development process using an iterative and incremental agile method		This methodology maximizes the team's ability to deliver quickly, respond to emerging requirements, and adapt to evolving technologies and changes in market conditions		
SDLC Phase Where It Is Used						
Analysis	Design	Development	Testing	Deployment	Maintenance	
X	X	X	X	X	X	
Roles (i.e. PM, Developer, etc.)			Scrum Master, Product Owner Development Team Member			
Related Methodologies, Best practices and Tools			Agile			
FOSS Communities Using This Methodology	OWASP	X	OpenSSL	N/A	Jenkins	N/A
	OpenStack	N/A	PIWIK	X	Eclipse	N/A
	Spring FW	N/A	Red Hat	N/A	Bitergia	N/A
	LibreOffice	N/A	Drupal	X	OwnCloud	N/A
	Debian	N/A	Apache Tomcat	?		N/A

M2. Methodology Name:							**Agile**
Use			**Objectives**			**Benefits**	
Software development			Manage the software development process using an iterative and incremental agile method			An agile method tailored to FOSS communities	
SDLC Phase Where It Is Used							
Analysis	Design	Development	Testing	Deployment	Maintenance		
X	X	X	X	X	X		
Roles (i.e. PM, Developer, etc.)					Business Analyst, System Architect, Test Architect, Project Manager, Tester, Developer		
Related Methodologies, Best Practices and Tools					N/A		
FOSS Communities Using This Methodology	OWASP	?	OpenSSL	N/A	Jenkins	N/A	
	OpenStack	N/A	PIWIK	X	Eclipse	N/A	
	Spring FW	N/A	Red Hat	N/A	Bitergia	N/A	
	LibreOffice	?	Drupal	?	OwnCloud	N/A	
	Debian	N/A	Apache Tomcat	?			

M3. Methodology Name:	**Kanban**

Use	**Objectives**	**Benefits**
Software development	Drive the software development process through well-defined, fixed phases, ensuring completeness and quality when an artefact moves to the next phase	Improve the quality and classification for the development of artefacts. Reduce bottlenecks

SDLC Phase Where It Is Used					
Analysis	Design	Development	Testing	Deployment	Maintenance
X	X	X	X	X	X

Roles (i.e. PM, Developer, etc.)	No existing roles. (Assistance from an Agile coach)
Related Methodologies, Best Practices and Tools	Agile

FOSS Communities Using This Methodology	OWASP	?	OpenSSL	N/A	Jenkins	N/A
	OpenStack	N/A	PIWIK	N/A	Eclipse	N/A
	Spring FW	N/A	Red Hat	N/A	Bitergia	N/A
	LibreOffice	?	Drupal	X	OwnCloud	N/A
	Debian	N/A	Apache Tomcat	?		

M4. Methodology Name:	**Waterfall**

Use	**Objectives**	**Benefits**
Software development	In this methodology, the software development activity is divided into different phases and each phase consists of series of tasks with different objectives. All phases are linked and executed in the right order	Easy to manage due to the rigidity of the model – each phase has specific deliverables and a review process. Phases are processed and completed one at a time and cannot overlap

SDLC Phase Where It Is Used					
Analysis	Design	Development	Testing	Deployment	Maintenance
X	X	X	X	X	X

Roles (i.e. PM, Developer, etc.)	Project Manager, Business Analyst, Architect, Developer, Tester, Release Manager
Related Methodologies, Best Practices and Tools	Waterfall

FOSS Communities Using This Methodology	OWASP	X	OpenSSL	N/A	Jenkins	N/A
	OpenStack	N/A	PIWIK	N/A	Eclipse	N/A
	Spring FW	N/A	Red Hat	N/A	Bitergia	N/A
	LibreOffice	?	Drupal	X	OwnCloud	N/A
	Debian	N/A	Apache Tomcat	?		

3.2. Best Practices Used by the Analysed FOSS Communities During the Software Development Lifecycle

BP1. Best Practice Name:		**Security in the Design**
Use	**Objectives**	**Benefits**
All components used in the application are included by validation, and analysed in terms of the provided security and functionality. The design includes a correct segregation for security reasons, including security analysis and threat assessment	Identifying all application components and possible risks. Defining segregation among application components and modules. Performing a deep study of application security and architecture	Good knowledge of application components. Effective security barriers according to application components and modules. Identification of possible application security flaws and risks

SDLC Phase Where It Is Used					
Analysis	Design	Development	Testing	Deployment	Maintenance
X	X	N/A	N/A	N/A	N/A

Roles (i.e. Analyst, Developer, Tester)	Security Architect, Analyst
Related Methodologies, Best Practices and Tools	
Related Technologies (i.e. Java)	

FOSS Communities Using This Best Practice	OWASP	X	OpenSSL	X	Jenkins	N/A
	OpenStack	X	PIWIK	N/A	Eclipse	N/A
	Spring FW	X	Red Hat	N/A	Bitergia	N/A
	LibreOffice	?	Drupal	N/A	OwnCloud	X
	Debian	N/A	Apache Tomcat	?		

BP2. Best Practice Name:	**Effective Authentication**

Use	**Objectives**	**Benefits**
All pages and resources require authentication. The authentication mechanism as well as the password recovery mechanism are strong enough to secure: • Encrypted transport of credentials • Enforcing password change and password strength • Implementation of server-side authentication controls that fail securely	Verifying the communications source. Verifying that only authorised users can access the application. Credentials are managed correctly during transport and storage	Application access is managed in a secure way, allowing entry only to authenticated users

SDLC Phase Where It Is Used					
Analysis	Design	Development	Testing	Deployment	Maintenance
N/A	X	X	N/A	N/A	N/A

Roles (i.e. Analyst, Developer, Tester)	Security Architect, Analyst, Developer
Related Methodologies, Best Practices and Tools	
Related Technologies (i.e. Java)	

FOSS Communities Using This Best Practice	OWASP	X	OpenSSL	N/A	Jenkins	X
	OpenStack	X	PIWIK	N/A	Eclipse	X
	Spring FW	X	Red Hat	X	Bitergia	N/A
	LibreOffice	?	Drupal	?	OwnCloud	X
	Debian	X	Apache Tomcat	X		

BP3. Best Practice Name:	**Secure Session Management**

Use	**Objectives**	**Benefits**
Use of a robust and tested session manager, resistant to common session attacks. Sessions must be created for each authentication/re-authentication process, and killed correctly after a log-out or time-out. For multiple sessions, the number of active sessions must be limited, allowing users to check active sessions and close them	Session management is secure enough, and resistant to common attacks. Multiple sessions can be managed easily. Sessions are killed securely minimizing the attack surface.	Sessions are managed securely, disallowing entry of unauthorised users

SDLC Phase Where It Is Used					
Analysis	Design	Development	Testing	Deployment	Maintenance
N/A	X	X	N/A	N/A	N/A

Roles (i.e. Analyst, Developer, Tester)	Security Architect, Analyst, Developer
Related Methodologies, Best Practices and Tools	
Related Technologies (i.e. Java)	

FOSS Communities Using This Best Practice	OWASP	X	OpenSSL	N/A	Jenkins	N/A
	OpenStack	X	PIWIK	N/A	Eclipse	X
	Spring FW	X	Red Hat	N/A	Bitergia	N/A
	LibreOffice	?	Drupal	?	OwnCloud	X
	Debian	X	Apache Tomcat	?		

BP4. Best Practice Name:	**Secure Access Control**

Use	**Objectives**	**Benefits**
A set of roles and privileges are in place for managing access to application resources, following the principle of "least privilege". In case of failure, it does so securely without allowing default access. Any attempt to access a resource can be logged, whether it is successful or not. To avoid unauthorised access to application resources, directory browsing has to be disabled	Users obtain access only to the resources allowed to them. Unauthorised users cannot access application resources. Any attempt to access resources can be tracked	Application resources are accessed only by authorised users

SDLC Phase Where It Is Used					
Analysis	Design	Development	Testing	Deployment	Maintenance
N/A	X	X	N/A	N/A	N/A

Roles (i.e. Analyst, Developer, Tester)	Security Architect, Analyst, Developer
Related Methodologies, Best Practices and Tools	
Related Technologies (i.e. Java)	

FOSS Communities Using This Best Practice						
	OWASP	X	OpenSSL	N/A	Jenkins	X
	OpenStack	X	PIWIK	N/A	Eclipse	X
	Spring FW	X	Red Hat	X	Bitergia	N/A
	LibreOffice	?	Drupal	?	OwnCloud	X
	Debian	X	Apache Tomcat	?		

BP5. Best Practice Name:	**Malicious Input Handling**

Use	**Objectives**	**Benefits**
The application should include input control mechanisms for avoiding common risks such as code injection (SQL, XML, OS commands, XSS, LDAP commands, etc.), and buffer overflows. All input must be validated, regardless of entry point. All displayed information should be validated too, to avoid execution of injected code	All application input is checked and sanitized according to its purpose. Data external to the application is not trusted, and handled as potentially malicious input	Application resources are accessed only by authorised users

SDLC Phase Where It Is Used					
Analysis	Design	Development	Testing	Deployment	Maintenance
N/A	X	X	N/A	N/A	N/A

Roles (i.e. Analyst, Developer, Tester)	Security Architect, Analyst, Developer
Related Methodologies, Best Practices and Tools	
Related Technologies (i.e. Java)	

FOSS Communities Using This Best Practice	OWASP	X	OpenSSL	X	Jenkins	N/A
	OpenStack	X	PIWIK	N/A	Eclipse	N/A
	Spring FW	X	Red Hat	N/A	Bitergia	N/A
	LibreOffice	?	Drupal	?	OwnCloud	N/A
	Debian	N/A	Apache Tomcat	?		

BP6. Best Practice Name:	**Cryptography**

Use	**Objectives**	**Benefits**
Cryptographic modules must fail in a secure way, disallowing oracle padding. The algorithms they provide have to be validated against FIPS140-2 or an equivalent standard. Cryptographic keys must be isolated from the cryptographic service consumer	Cryptographic modules should fail securely. Secure access to keys. Only good quality random number generator should be used.	Cryptographic modules are strong enough to support all cryptographic needs of the application

SDLC Phase Where It Is Used					
Analysis	Design	Development	Testing	Deployment	Maintenance
N/A	X	X	N/A	N/A	N/A

Roles (i.e. Analyst, Developer, Tester)	Security Architect, Analyst, Developer
Related Methodologies, Best Practices and Tools	
Related Technologies (i.e. Java)	

FOSS Communities Using This Best Practice						
	OWASP	X	OpenSSL	X	Jenkins	N/A
	OpenStack	X	PIWIK	N/A	Eclipse	N/A
	Spring FW	X	Red Hat	N/A	Bitergia	N/A
	LibreOffice	?	Drupal	?	OwnCloud	X
	Debian	X	Apache Tomcat	?		

BP7. Best Practice Name:	**Data Protection**

Use	**Objectives**	**Benefits**
All forms containing sensitive information should have client-side caching disabled. All sensitive data must be sent to the server in the HTTP message body or headers. The application sets any appropriate anti-caching headers according to application risk. Data storage on the client-side does not retain sensitive information	Data should be protected from unauthorised access. Data should be protected from malicious modification by unauthorised users. Data should be available to authorised users upon request	Data managed by the application can be used correctly, and accessed when requested by authorised users.

SDLC Phase Where It Is Used					
Analysis	Design	Development	Testing	Deployment	Maintenance
N/A	X	X	N/A	N/A	N/A

Roles (i.e. Analyst, Developer, Tester)	Security Architect, Analyst, Developer
Related Methodologies, Best Practices and Tools	
Related Technologies (i.e. Java)	

FOSS Communities Using This Best Practice	OWASP	X	OpenSSL	X	Jenkins	X
	OpenStack	X	PIWIK	N/A	Eclipse	N/A
	Spring FW	X	Red Hat	N/A	Bitergia	N/A
	LibreOffice	?	Drupal	?	OwnCloud	X
	Debian	N/A	Apache Tomcat	?		

BP3. Best Practice Name:	**Secure Communication**

Use	**Objectives**	**Benefits**
Use certificates for Transport Layer Security (TLS) from trusted CAs. Ensure that TLS fails securely and only the most secure algorithm is used. HTTP Strict Transport Security headers have to be included in all requests. Verify that the TLS configuration is aligned with current leading practises	TLS is used in any communication where sensitive data is transmitted. Strong algorithms and ciphers are used in any communication	Communications are secure enough, disallowing information disclosure to unauthorised users

SDLC Phase Where It Is Used					
Analysis	Design	Development	Testing	Deployment	Maintenance
N/A	X	X	N/A	N/A	N/A

Roles (i.e. Analyst, Developer, Tester)	Security Architect, Analyst, Developer
Related Methodologies, Best Practices and Tools	
Related Technologies (i.e. Java)	

FOSS Communities Using This Best Practice	OWASP	X	OpenSSL	X	Jenkins	X
	OpenStack	X	PIWIK	N/A	Eclipse	N/A
	Spring FW	X	Red Hat	N/A	Bitergia	N/A
	LibreOffice	?	Drupal	?	OwnCloud	N/A
	Debian	N/A	Apache Tomcat	?		

BP9. Best Practice Name:	**HTTP Security Configuration**

Use	**Objectives**	**Benefits**
The application only accepts a set of HTTP request methods (PUT and POST), blocking unused methods. Every HTTP response contains secure configuration about content type (e.g. safe character set such as UTF-8). HTTP headers do not expose version information of system components	The application server is hardened by the default configuration. HTTP responses contain a safe character set in the content type header	HTTP security is improved considerably, minimizing the attack surface.

SDLC Phase Where It Is Used					
Analysis	Design	Development	Testing	Deployment	Maintenance
N/A	X	X	N/A	N/A	N/A

Roles (i.e. Analyst, Developer, Tester)	Security Architect, Analyst, Developer, System Administration
Related Methodologies, Best Practices and Tools	
Related Technologies (i.e. Java)	

FOSS Communities Using This Best Practice	OWASP	X	OpenSSL	X	Jenkins	X
	OpenStack	X	PIWIK	N/A	Eclipse	N/A
	Spring FW	N/A	Red Hat	N/A	Bitergia	N/A
	LibreOffice	?	Drupal	?	OwnCloud	X
	Debian	N/A	Apache Tomcat	?		

BP10. Best Practice Name:	**Malicious Controls**

Use	**Objectives**	**Benefits**
All malicious activity should be containerized to delay and stop attackers. Code reviews will be conducted for detection of back doors, Easter eggs and logic flaws	Detection of malicious activity is managed securely; it is properly isolated to avoid affecting the remaining application	Malicious controls reduce the attack surface, and provide mechanisms to detect and prevent such future actions

SDLC Phase Where It Is Used					
Analysis	Design	Development	Testing	Deployment	Maintenance
N/A	X	X	N/A	N/A	N/A

Roles (i.e. Analyst, Developer, Tester)	Security Architect, Analyst, Developer, System Administration
Related Methodologies, Best Practices and Tools	
Related Technologies (i.e. Java)	

FOSS Communities Using This Best Practice	OWASP	X	OpenSSL	X	Jenkins	N/A
	OpenStack	N/A	PIWIK	N/A	Eclipse	N/A
	Spring FW	X	Red Hat	N/A	Bitergia	N/A
	LibreOffice	?	Drupal	?	OwnCloud	N/A
	Debian	N/A	Apache Tomcat	?		

BP11. Best Practice Name:			**Secure Business Logic**			
Use			**Objectives**		**Benefits**	
The application should process any business logic in an orderly manner, avoiding bot activity. Business limits should be set, with alerting mechanisms and automated responses to automated or unusual attacks			Sequential business logic flow. Avoid bot activity		Enforcement of expected execution of use cases in order to avoid odd behaviours and bot activity	
SDLC Phase Where It Is Used						
Analysis	Design	Development	Testing	Deployment	Maintenance	
N/A	X	X	N/A	N/A	N/A	
Roles (i.e. Analyst, Developer, Tester)			Security Architect, Analyst, Developer			
Related Methodologies, Best Practices and Tools						
Related Technologies (i.e. Java)						
FOSS Communities Using This Best Practice	OWASP	X	OpenSSL	N/A	Jenkins	N/A
	OpenStack	N/A	PIWIK	N/A	Eclipse	N/A
	Spring FW	N/A	Red Hat	N/A	Bitergia	N/A
	LibreOffice	?	Drupal	?	OwnCloud	X
	Debian	N/A	Apache Tomcat	?		

BP12. Best Practice Name:	**Secure File and Resource Management**

Use	**Objectives**	**Benefits**
The application should prevent the upload of malicious files by validating the file types (i.e. if the application is expecting a .pdf file, then a .pdf file is uploaded) Additionally, use an antivirus scanner, disable direct code execution from external files, avoid reflexion capabilities (i.e. a code injection embedded in a comment section inside a blog is executed when another user access the blog). Avoid using technologies not supported natively via W3C browser standards.	Untrusted file data have to be managed securely. Data from untrusted sources should be stored outside the web root environment and have limited permissions	Secure file management within the application. Minimisation of attack surface.

SDLC Phase Where It Is Used					
Analysis	Design	Development	Testing	Deployment	Maintenance
N/A	X	X	N/A	N/A	N/A

Roles (i.e. Analyst, Developer, Tester)	Security Architect, Analyst, Developer
Related Methodologies, Best Practices and Tools	
Related Technologies (i.e. Java)	

FOSS Communities Using This Best Practice	OWASP	X	OpenSSL	N/A	Jenkins	?
	OpenStack	N/A	PIWIK	N/A	Eclipse	?
	Spring FW	N/A	Red Hat	N/A	Bitergia	?
	LibreOffice	?	Drupal	?	OwnCloud	X
	Debian	N/A	Apache Tomcat	?		

BP13. Best Practice Name:	**Secure Mobile Application**

Use	**Objectives**	**Benefits**
Verify that any token, secret key or password is generated automatically. Any data stored in shared resources should be non-sensitive data. Sensitive data should be secured, even if it is stored in protected areas. Application sensitive code should be laid out randomly in memory. UDID or IMEI number should be used as authentication tokens	All server-side controls (API, Web Service) should have the same security level with those deployed on the device. Sensitive information stored on the device should be managed securely (storage and transmission)	The mobile application can manage and transmit data to a server in a secure way

SDLC Phase Where It Is Used					
Analysis	Design	Development	Testing	Deployment	Maintenance
N/A	X	X	N/A	N/A	N/A

Roles (i.e. Analyst, Developer, Tester)	Security Architect, Analyst, Developer
Related Methodologies, Best Practices and Tools	
Related Technologies (i.e. Java)	

FOSS Communities Using This Best Practice	OWASP	X	OpenSSL	N/A	Jenkins	N/A
	OpenStack	X	PIWIK	N/A	Eclipse	N/A
	Spring FW	N/A	Red Hat	N/A	Bitergia	N/A
	LibreOffice	?	Drupal	?	OwnCloud	X
	Debian	N/A	Apache Tomcat	?		

BP14. Best Practice Name:	**Secure Web Services**

Use	**Objectives**	**Benefits**
Client and server have same encoding style. XML or JSON schema is in place and verified before accepting input. All input has a limited size. Verified session is based on authentication and authorization. Avoid the use of static “API keys” and similar. Rest service has to be protected from CRSF (Cross-Site Request Forgery)	Web services (RESTful or SOAP) should have secure authentication mechanisms, secure session management, and secure authorization. All parameters sent have to be validated when they are transmitted from lower trust levels to higher trust levels. Basic interoperability of SOAP web services layer to promote API use	Secure data exchange between web services. Secure entry points as far as web services are concerned

SDLC Phase Where It Is Used					
Analysis	Design	Development	Testing	Deployment	Maintenance
N/A	X	X	N/A	N/A	N/A

Roles (i.e. Analyst, Developer, Tester)	Security Architect, Analyst, Developer
Related Methodologies, Best Practices and Tools	
Related Technologies (i.e. Java)	

FOSS Communities Using This Best Practice	OWASP	X	OpenSSL	X	Jenkins	X
	OpenStack	X	PIWIK	N/A	Eclipse	N/A
	Spring FW	N/A	Red Hat	N/A	Bitergia	N/A
	LibreOffice	?	Drupal	?	OwnCloud	X
	Debian	N/A	Apache Tomcat	?		

BP15. Best Practice Name:			**Secure Configuration**			
Use			**Objectives**		**Benefits**	
All components should be up-to-date, with secure configurations, removing unneeded settings and folders. Communication with the component application has to be encrypted and authenticated using accounts with least possible privileges. The application should be isolated to avoid compromising other applications			Libraries and platforms should be updated. Secure-by-default configuration. Enough hardening to avoid security breaches of underlying systems, in case of changes in the default configuration		Application components should be more secure, with last known issues corrected. In case of a security incident in one of the components, this will not be propagated	
SDLC Phase Where It Is Used						
Analysis	Design	Development	Testing	Deployment	Maintenance	
N/A	X	X	N/A	N/A	X	
Roles (i.e. Analyst, Developer, Tester)			Security Architect, Analyst, Developer, System Administrator			
Related Methodologies, Best Practices and Tools						
Related Technologies (i.e. Java)						
FOSS Communities Using This Best Practice	OWASP	X	OpenSSL	X	Jenkins	X
	OpenStack	N/A	PIWIK	N/A	Eclipse	N/A
	Spring FW	X	Red Hat	N/A	Bitergia	N/A
	LibreOffice	?	Drupal	?	OwnCloud	N/A
	Debian	N/A	Apache Tomcat	?		

BP16. Best Practice Name:			**Threat Assessment**			
Use			**Objectives**		**Benefits**	
Creating application-specific threat models and attacker profiles from the software architecture. Designing abuse case models, adding countermeasures and evaluating explicitly third-party components' risks			This analysis is centred on identification and understanding of the software risk under development		Awareness of possible security flaws in the application. Knowledge of what security controls to deploy in the application	
SDLC Phase Where It Is Used						
Analysis	Design	Development	Testing	Deployment	Maintenance	
N/A	X	N/A	N/A	N/A	N/A	
Roles (i.e. Analyst, Developer, Tester)			Security Architect, Analyst			
Related Methodologies, Best Practices and Tools						
Related Technologies (i.e. Java)						
FOSS Communities Using This Best Practice	OWASP	X	OpenSSL	X	Jenkins	N/A
	OpenStack	X	PIWIK	N/A	Eclipse	N/A
	Spring FW	N/A	Red Hat	N/A	Bitergia	N/A
	LibreOffice	?	Drupal	?	OwnCloud	N/A
	Debian	N/A	Apache Tomcat	?		

BP17. Best Practice Name:		**Security Requirements**
Use	**Objectives**	**Benefits**
Gathering security requirements explicitly during the requirement analysis phase. Identifying security guidelines to follow. Designing an access control matrix for application resources and privileges	Adding security at the beginning of the Software Development Life Cycle (SDLC)	Cost of security implementation is significantly decreased. Application functionality is designed in a secure manner

SDLC Phase Where It Is Used					
Analysis	Design	Development	Testing	Deployment	Maintenance
X	X	N/A	N/A	N/A	N/A

Roles (i.e. Analyst, Developer, Tester)	Security Architect, Analyst
Related Methodologies, Best Practices and Tools	
Related Technologies (i.e. Java)	

FOSS Communities Using This Best Practice	OWASP	X	OpenSSL	X	Jenkins	N/A
	OpenStack	X	PIWIK	N/A	Eclipse	N/A
	Spring FW	X	Red Hat	X	Bitergia	N/A
	LibreOffice	X	Drupal	?	OwnCloud	X
	Debian	N/A	Apache Tomcat	?		

BP18. Best Practice Name:	**Secure Architecture**

Use	**Objectives**	**Benefits**
Built and maintain a list of recommended software frameworks. Apply security principles to design, identify security design patterns, and promote security services and infrastructure	Insert proactive security guidance into software design phase. Implement known secure services and security by default in designs	Robust and security-oriented application architecture

SDLC Phase Where It Is Used					
Analysis	Design	Development	Testing	Deployment	Maintenance
N/A	X	N/A	N/A	N/A	N/A

Roles (i.e. Analyst, Developer, Tester)	Security Architect, Analyst
Related Methodologies, Best Practices and Tools	
Related Technologies (i.e. Java)	

FOSS Communities Using This Best Practice	OWASP	X	OpenSSL	X	Jenkins	N/A
	OpenStack	X	PIWIK	N/A	Eclipse	N/A
	Spring FW	N/A	Red Hat	N/A	Bitergia	N/A
	LibreOffice	?	Drupal	?	OwnCloud	N/A
	Debian	N/A	Apache Tomcat	X		

BP19. Best Practice Name:	**Design Review**

Use	**Objectives**	**Benefits**
Identifying software attack vectors in the application. Matching of the application design against the security requirements. Doing data-flow diagrams for sensitive resources. Inspecting deep security mechanisms. Implementing design reviews as a release gate	Design a security review. Understand the application's attack vectors. Understanding the application's security-sensitive processes	Awareness of the application's perimeter architecture. Knowledge of all sensitive and security related operation implementations. Security is checked in each release

SDLC Phase Where It Is Used					
Analysis	Design	Development	Testing	Deployment	Maintenance
N/A	N/A	N/A	X	X	N/A

Roles (i.e. Analyst, Developer, Tester)	Security Architect, Analyst, Developer
Related Methodologies, Best Practices and Tools	
Related Technologies (i.e. Java)	

FOSS Communities Using This Best Practice						
	OWASP	X	OpenSSL	X	Jenkins	N/A
	OpenStack	N/A	PIWIK	N/A	Eclipse	X
	Spring FW	N/A	Red Hat	N/A	Bitergia	N/A
	LibreOffice	?	Drupal	?	OwnCloud	N/A
	Debian	N/A	Apache Tomcat	?		

BP20. Best Practice Name:		**Code Review**
Use	**Objectives**	**Benefits**
Create review checklists form known security requirements and high-risk code. Add code analysis in the development process, and use an automated code analysis tool. Establish code review as a release gate	Apply security checkpoints to developed code. Insert formal code reviews during the development process	Better and more robust code is produced. Possible security issues arise and are corrected during development

SDLC Phase Where It Is Used					
Analysis	Design	Development	Testing	Deployment	Maintenance
N/A	N/A	X	X	N/A	N/A

Roles (i.e. Analyst, Developer, Tester)	Analyst, Developer, Security Tester
Related Methodologies, Best Practices and Tools	
Related Technologies (i.e. Java)	

FOSS Communities Using This Best Practice	OWASP	X	OpenSSL	X	Jenkins	N/A
	OpenStack	X	PIWIK	X	Eclipse	X
	Spring FW	X	Red Hat	X	Bitergia	X
	LibreOffice	X	Drupal	X	OwnCloud	X
	Debian	X	Apache Tomcat	X		

BP21. Best Practice Name:			**Security Testing**		
Use		**Objectives**		**Benefits**	
Design test cases from known security requirements. Conduct penetration testing on software releases. Integrate security testing into the development process. Establish release gates for security testing		Some testing cases are explicit security testing cases. Security checkpoints before releasing a new version		More robust and secure software is developed. Possible security flaws arise before deployment in production	
SDLC Phase Where It Is Used					
Analysis	Design	Development	Testing	Deployment	Maintenance
N/A	N/A	N/A	X	N/A	N/A
Roles (i.e. Analyst, Developer, Tester)			Security Tester		
Related Methodologies, Best Practices and Tools					
Related Technologies (i.e. Java)					

FOSS Communities Using This Best Practice	OWASP	X	OpenSSL	X	Jenkins	X
	OpenStack	X	PIWIK	N/A	Eclipse	N/A
	Spring FW	X	Red Hat	N/A	Bitergia	N/A
	LibreOffice	?	Drupal	?	OwnCloud	N/A
	Debian	N/A	Apache Tomcat	?		

BP22. Best Practice Name:		**Vulnerability Management**
Use	**Objectives**	**Benefits**
Create a security response team to deal with security issues. Establish a security process to face security issues. Collect per-incident metrics, and adopt a security issue disclosure process. Identify points of contact for security issues	Create a unit or working group to deal with security issues. Create mechanisms to effectively respond to incidents and exchange information with stakeholders	Effective way to handle software vulnerabilities and manage security issue information

SDLC Phase Where It Is Used					
Analysis	Design	Development	Testing	Deployment	Maintenance
N/A	N/A	N/A	N/A	N/A	X

Roles (i.e. Analyst, Developer, Tester)	Security Analyst, Developer
Related Methodologies, Best Practices and Tools	
Related Technologies (i.e. Java)	

FOSS Communities Using This Best Practice	OWASP	X	OpenSSL	X	Jenkins	X
	OpenStack	X	PIWIK	X	Eclipse	N/A
	Spring FW	N/A	Red Hat	X	Bitergia	N/A
	LibreOffice	X	Drupal	X	OwnCloud	X
	Debian	X	Apache Tomcat	X		

BP23. Best Practice Name:		**Operational Enablement**
Use	**Objectives**	**Benefits**
Document procedures for typical application alerts. Maintain formal operational security guides. Create change management procedures for each release	Provide useful security information to the operations team	Operations teams as part of security defences

SDLC Phase Where It Is Used					
Analysis	Design	Development	Testing	Deployment	Maintenance
N/A	N/A	N/A	N/A	N/A	X

Roles (i.e. Analyst, Developer, Tester)	System Administrator, Developer, Analyst
Related Methodologies, Best Practices and Tools	
Related Technologies (i.e. Java)	

FOSS Communities Using This Best Practice	OWASP	X	OpenSSL	N/A	Jenkins	N/A
	OpenStack	X	PIWIK	X	Eclipse	N/A
	Spring FW	N/A	Red Hat	N/A	Bitergia	N/A
	LibreOffice	?	Drupal	X	OwnCloud	N/A
	Debian	X	Apache Tomcat	X		

BP24. Best Practice Name:		**Different Security Teams**
Use	**Objectives**	**Benefits**
Different security teams work on the project. The first one supports software development by creating security guides, conducting threat analysis, and providing security tools. The second one is specifically in charge of vulnerability management	Security awareness during development. Provide a good practise guide. Provide security and support tools	Better code security. Persistent use of security tools

SDLC Phase Where It Is Used					
Analysis	Design	Development	Testing	Deployment	Maintenance
N/A	N/A	N/A	X	X	X

Roles (i.e. Analyst, Developer, Tester)	Security Experts
Related Methodologies, Best Practices and Tools	
Related Technologies (i.e. Java)	

FOSS Communities Using This Best Practice	OWASP	?	OpenSSL	N/A	Jenkins	N/A
	OpenStack	X	PIWIK	N/A	Eclipse	N/A
	Spring FW	N/A	Red Hat	X	Bitergia	N/A
	LibreOffice	X	Drupal	?	OwnCloud	X
	Debian	X	Apache Tomcat	?		

BP25. Best Practice Name:			**Role-Based Authorisation**			
Use			**Objectives**		**Benefits**	
Authorisation based on roles that provide an easy way to grant privileges to users according to different user profiles. This allows centralized management of user privileges			Centralisation of user privileges according to user profiles		Access to application resources is granted in a secure way. Users only have access to the resources defined in the user profile	
SDLC Phase Where It Is Used						
Analysis	Design	Development	Testing	Deployment	Maintenance	
X	X	X	N/A	N/A	N/A	
Roles (i.e. Analyst, Developer, Tester)			Analyst, Architect, Developer			
Related Methodologies, Best Practices and Tools						
Related Technologies (i.e. Java)						
FOSS Communities Using This Best Practice	OWASP	?	OpenSSL	X	Jenkins	N/A
	OpenStack	X	PIWIK	X	Eclipse	X
	Spring FW	X	Red Hat	N/A	Bitergia	N/A
	LibreOffice	?	Drupal	X	OwnCloud	N/A
	Debian	N/A	Apache Tomcat	X		

BP26. Best Practice Name:			**Standard Cryptographic Module**			
Use		**Objectives**		**Benefits**		
Use of a robust and tested cryptographic module, instead of creating a customised one		Robust encryption algorithm implementation		Secure application communications. Tested library for critical information management		
SDLC Phase Where It Is Used						
Analysis	Design	Development	Testing	Deployment	Maintenance	
N/A	X	X	N/A	N/A	N/A	
Roles (i.e. Analyst, Developer, Tester)			Analyst, Developer, Architect			
Related Methodologies, Best Practices and Tools			OpenSSL			
Related Technologies (i.e. Java)						
FOSS Communities Using This Best Practice	OWASP	?	OpenSSL	X	Jenkins	N/A
	OpenStack	N/A	PIWIK	N/A	Eclipse	N/A
	Spring FW	N/A	Red Hat	N/A	Bitergia	N/A
	LibreOffice	?	Drupal	?	OwnCloud	N/A
	Debian	N/A	Apache Tomcat	X		

BP27. Best Practice Name:			**Well-Tested Base Technology**			
Use		**Objectives**		**Benefits**		
Use a robust and mature underlying technology for application development that has been tested in many production environments, and is widely in use		Minimize zero-day vulnerability risks. Active development of the base technology provides new security functions and patches on an as-needed basis		Takes advantage of any prior experience regarding security of the underlying technology. Reduces the number of possible security flaws in the base technology		
SDLC Phase Where It Is Used						
Analysis	Design	Development	Testing	Deployment	Maintenance	
N/A	X	X	N/A	N/A	N/A	
Roles (i.e. Analyst, Developer, Tester)			Analyst, Developer, Architect			
Related Methodologies, Best Practices and Tools						
Related Technologies (i.e. Java)			Java , PHP			
FOSS Communities Using This Best Practice	OWASP	?	OpenSSL	N/A	Jenkins	N/A
	OpenStack	N/A	PIWIK	X	Eclipse	N/A
	Spring FW	N/A	Red Hat	?	Bitergia	N/A
	LibreOffice	?	Drupal	X	OwnCloud	N/A
	Debian	X	Apache Tomcat	X		

BP28. Best Practice Name:			**Continuous Testing**			
Use		**Objectives**		**Benefits**		
Release management		Execute automated tests as part of the software delivery pipeline		The development team can prevent problems from progressing to the next stage of SDLC. Reduce required time and effort in fixing defects		
SDLC Phase Where It Is Used						
Analysis	Design	Development	Testing	Deployment	Maintenance	
N/A	N/A	X	N/A	N/A	X	
Roles (i.e. Analyst, Developer, Tester)			Tester, Operations			
Related Methodologies, Best Practices and Tools			Continuous delivery, continuous deployment			
Related Technologies (i.e. Java)						
FOSS Communities Using This Best Practice	OWASP	?	OpenSSL	X	Jenkins	X
	OpenStack	X	PIWIK	X	Eclipse	X
	Spring FW	X	Red Hat	?	Bitergia	N/A
	LibreOffice	X	Drupal	X	OwnCloud	X
	Debian	X	Apache Tomcat			

BP29. Best Practice Name:						**Automation Testing**	
Use			**Objectives**		**Benefits**		
Validation and testing			Automatically execute written tests without manual intervention		Increase effectiveness, efficiency and coverage of software testing		
SDLC Phase Where It Is Used							
Analysis	Design	Development	Testing	Deployment	Maintenance		
N/A	N/A	N/A	X	X	N/A		
Roles (i.e. Analyst, Developer, Tester)					Developer, Tester		
Related Methodologies, Best Practices and Tools					Validation and testing, unit testing, integration testing, functional testing, non-functional testing, regression testing		
Related Technologies (i.e. Java)							
FOSS Communities Using This Best Practice	OWASP	X	OpenSSL	N/A	Jenkins	X	
	OpenStack	X	PIWIK	X	Eclipse	X	
	Spring FW	X	Red Hat	N/A	Bitergia	N/A	
	LibreOffice	X	Drupal	X	OwnCloud	X	
	Debian	X	Apache Tomcat	N/A			

BP30. Best Practice Name:	**Third-Party Testing**

Use	**Objectives**	**Benefits**
Application testing conducted by an external team of security experts	Find possible security flaws and misconfigurations	Detect security vulnerabilities in the application Detect configuration errors in the application and application environment

SDLC Phase Where It Is Used					
Analysis	Design	Development	Testing	Deployment	Maintenance
N/A	N/A	N/A	X	X	X

Roles (i.e. Analyst, Developer, Tester)	Security Auditor, System Administrator
Related Methodologies, Best Practices and Tools	
Related Technologies (i.e. Java)	

FOSS Communities Using This Best Practice	OWASP	?	OpenSSL	N/A	Jenkins	N/A
	OpenStack	N/A	PIWIK	N/A	Eclipse	N/A
	Spring FW	N/A	Red Hat	N/A	Bitergia	N/A
	LibreOffice	?	Drupal	X	OwnCloud	X
	Debian	N/A	Apache Tomcat	?		

BP31. Best Practice Name:					**Release Planning**
Use		**Objectives**			**Benefits**
Release management		Plan and schedule releases and define their scope			The release is planned in advance
SDLC Phase Where It Is Used					
Analysis	Design	Development	Testing	Deployment	Maintenance
N/A	N/A	N/A	N/A	X	N/A
Roles (i.e. Analyst, Developer, Tester)					Project Manager, Operations
Related Methodologies, Best Practices and Tools					Release Management
Related Technologies (i.e. Java)					

FOSS Communities Using This Best Practice	OWASP	X	OpenSSL	X	Jenkins	X
	OpenStack	X	PIWIK	N/A	Eclipse	N/A
	Spring FW	X	Red Hat	X	Bitergia	N/A
	LibreOffice	X	Drupal	X	OwnCloud	X
	Debian	X	Apache Tomcat	X		

BP32. Best Practice Name:					**Security Incident Management**
Use		**Objectives**			**Benefits**
The security incident management is part of the security plan, which explains how to respond to incidents		In case of a security incident, clearly define the response action and contact person (in key staff)			Provides an effective mechanism to mitigate the possible impact of a security incident
SDLC Phase Where It Is Used					
Analysis	Design	Development	Testing	Deployment	Maintenance
N/A	N/A	N/A	N/A	X	X
Roles (i.e. Analyst, Developer, Tester)				System Administrator, Project Manager, CISO or LISO	
Related Methodologies, Best Practices and Tools					
Related Technologies (i.e. Java)					

FOSS Communities Using This Best Practice	OWASP	X	OpenSSL	X	Jenkins	N/A
	OpenStack	X	PIWIK	X	Eclipse	X
	Spring FW	X	Red Hat	X	Bitergia	X
	LibreOffice	?	Drupal	X	OwnCloud	X
	Debian	X	Apache Tomcat	X		

BP33. Best Practice Name:			**Proactive Problem Identification**		
Use		**Objectives**		**Benefits**	
Problem management		Improve the application quality by identifying bugs and vulnerabilities in advance		Advance issue addressing that allow quick fixes or workaround implementations	
SDLC Phase Where It Is Used					
Analysis	Design	Development	Testing	Deployment	Maintenance
N/A	N/A	N/A	N/A	N/A	X
Roles (i.e. Analyst, Developer, Tester)			IT Team		
Related Methodologies, Best Practices and Tools			ITIL [1], Problem management		
Related Technologies (i.e. Java)					

FOSS Communities Using This Best Practice	OWASP	X	OpenSSL	X	Jenkins	X
	OpenStack	X	PIWIK	X	Eclipse	X
	Spring FW	X	Red Hat	X	Bitergia	N/A
	LibreOffice	X	Drupal	X	OwnCloud	X
	Debian	X	Apache Tomcat	X		

BP34. Best Practice Name:			**Security Incident Notification**		
Use		**Objectives**		**Benefits**	
Security incidents are communicated to users, informing them about the impact and suggesting actions for containment or resolution (i.e. password change)		Effective communication with users on security issues. Provide some security awareness information to users		Users believe that security is managed correctly. Users can perform mitigation actions if a security incident affects them	
SDLC Phase Where It Is Used					
Analysis	Design	Development	Testing	Deployment	Maintenance
N/A	N/A	N/A	N/A	N/A	X
Roles (i.e. Analyst, Developer, Tester)			Project Manager		
Related Methodologies, Best Practices and Tools					
Related Technologies (i.e. Java)					

FOSS Communities Using This Best Practice	OWASP	X	OpenSSL	X	Jenkins	N/A
	OpenStack	X	PIWIK	X	Eclipse	X
	Spring FW	X	Red Hat	X	Bitergia	X
	LibreOffice	X	Drupal	X	OwnCloud	X
	Debian	X	Apache Tomcat	X		

BP35. Best Practice Name:			**Getting Involved in the Community**		
Use		**Objectives**		**Benefits**	
Product building		Follow development trends, help other community members, share ideas		Find help, ideas, and solutions	
SDLC Phase Where It Is Used					
Analysis	Design	Development	Testing	Deployment	Maintenance
X	X	X	X	X	X
Roles (i.e. Analyst, Developer, Tester)			Analyst, Developer, Tester, QA		
Related Methodologies, Best Practices and Tools					
Related Technologies (i.e. Java)					

FOSS Communities Using This Best Practice	OWASP	X	OpenSSL	X	Jenkins	X
	OpenStack	X	PIWIK	X	Eclipse	X
	Spring FW	X	Red Hat	N/A	Bitergia	X
	LibreOffice	X	Drupal	?	OwnCloud	X
	Debian	X	Apache Tomcat	?		

BP36. Best Practice Name:			**Developer Community**		
Use		**Objectives**		**Benefits**	
Product building		Developers can contribute to the core or create plugins and modules		Align with the guidelines and procedures established in the community. Better knowledge of the documentation for HTTP, JavaScript and PHP API as used in the projects	
SDLC Phase Where It Is Used					
Analysis	Design	Development	Testing	Deployment	Maintenance
N/A	N/A	X	N/A	N/A	N/A
Roles (i.e. Analyst, Developer, Tester)			Developer		
Related Methodologies, Best Practices and Tools					
Related Technologies (i.e. Java)					

FOSS Communities Using This Best Practice	OWASP	X	OpenSSL	X	Jenkins	X
	OpenStack	X	PIWIK	X	Eclipse	X
	Spring FW	X	Red Hat	X	Bitergia	N/A
	LibreOffice	X	Drupal	X	OwnCloud	X
	Debian	X	Apache Tomcat	X		

BP37. Best Practice Name:	**API Documentation and Comments Best Practices and Standards**

Use	**Objectives**	**Benefits**
Development / Coding Standards	Proper documentation and commenting within the code	The documentation can be parsed and displayed by automated tools according to well-defined rules, and is also comprehensible to programmers. Integrated Development Environments (IDEs) can work seamlessly with the code and documentation

SDLC Phase Where It Is Used					
Analysis	Design	Development	Testing	Deployment	Maintenance
N/A	N/A	X	N/A	N/A	N/A

Roles (i.e. Analyst, Developer, Tester)	Developer, QA
Related Methodologies, Best Practices and Tools	
Related Technologies (i.e. Java)	

FOSS Communities Using This Best Practice	OWASP	X	OpenSSL	X	Jenkins	X
	OpenStack	X	PIWIK	X	Eclipse	N/A
	Spring FW	X	Red Hat	N/A	Bitergia	N/A
	LibreOffice	?	Drupal	?	OwnCloud	X
	Debian	X	Apache Tomcat	X		

BP38. Best Practice Name:	**Coding Standards and Best Practices**

Use	**Objectives**	**Benefits**
Development / Coding Standards	The Coding Standards apply to code within the core and its plugins. They provide a set of rules for how code should be formatted, guidelines for naming conventions, and the location of files. They contribute to code in a way that facilitates code review. They distribute contributions in small chunks. They include tests	They ensure code consistency throughout the project and make it easy for developers to understand other developers' code. Source code tends to appear as if it was written by the same developer

SDLC Phase Where It Is Used					
Analysis	Design	Development	Testing	Deployment	Maintenance
N/A	N/A	X	N/A	N/A	N/A

Roles (i.e. Analyst, Developer, Tester)	Developer, QA
Related Methodologies, Best Practices and Tools	
Related Technologies (i.e. Java)	

FOSS Communities Using This Best Practice	OWASP	X	OpenSSL	X	Jenkins	N/A
	OpenStack	X	PIWIK	X	Eclipse	X
	Spring FW	X	Red Hat	X	Bitergia	N/A
	LibreOffice	?	Drupal	X	OwnCloud	X
	Debian	X	Apache Tomcat	?		

BP39. Best Practice Name:			**Write Optimized and Performant Code**		
Use		**Objectives**		**Benefits**	
Development / Programming		Keep functions short and clear. Reuse existing code Write code that handles all possible scenarios. Write efficient code that will scale well		Smaller functions doing single tasks are easier to test and reuse. Code reusability reduces maintenance costs. Correct code introduces less bugs. Efficient code helps to improve application performance	
SDLC Phase Where It Is Used					
Analysis	Design	Development	Testing	Deployment	Maintenance
N/A	N/A	X	X	N/A	N/A
Roles (i.e. Analyst, Developer, Tester)		Developer, QA			
Related Methodologies, Best Practices and Tools					
Related Technologies (i.e. Java)					

FOSS Communities Using This Best Practice	OWASP	X	OpenSSL	X	Jenkins	X
	OpenStack	X	PIWIK	X	Eclipse	X
	Spring FW	X	Red Hat	X	Bitergia	N/A
	LibreOffice	?	Drupal	X	OwnCloud	X
	Debian	N/A	Apache Tomcat	?		

BP40. Best Practice Name:	**Write Secure Database Queries**

Use	**Objectives**	**Benefits**
Development / Writing secure code	Use prepared SQL statements to issue database queries; guard against SQL injection attacks	Prevent SQL injection attacks. DB APIs and database queries can be ACL compliant and become protected against several vulnerabilities. Database independent code can also be written

SDLC Phase Where It Is Used					
Analysis	Design	Development	Testing	Deployment	Maintenance
N/A	N/A	X	X	N/A	N/A

Roles (i.e. Analyst, Developer, Tester)	Developer, QA, Tester
Related Methodologies, Best Practices and Tools	
Related Technologies (i.e. Java)	

FOSS Communities Using This Best Practice	OWASP	X	OpenSSL	X	Jenkins	N/A
	OpenStack	N/A	PIWIK	X	Eclipse	N/A
	Spring FW	X	Red Hat	N/A	Bitergia	N/A
	LibreOffice	?	Drupal	X	OwnCloud	X
	Debian	X	Apache Tomcat	?		

BP41. Best Practice Name:	**Write Secure Code**

Use	**Objectives**	**Benefits**
Development / Writing secure code	Use specific functions and check that they handle input and output properly. Use the secure guidelines to build forms. Avoid using data directly from form-submitted fields (GET/POST) without checking or filtering them first. Do not use eval() to parse PHP code from user-entered text	Filtering output resulting from direct user input prevents cross site scripting attacks. Create forms in a safe way to avoid cross-site request forgeries (CSRF). Evaluating PHP code in user-entered text is a security risk (the PHP input might contain malicious code) and should be avoided

SDLC Phase Where It Is Used					
Analysis	Design	Development	Testing	Deployment	Maintenance
N/A	N/A	X	X	N/A	N/A

Roles (i.e. Analyst, Developer, Tester)	Developer, QA, Tester
Related Methodologies, Best Practices and Tools	
Related Technologies (i.e. Java)	

FOSS Communities Using This Best Practice	OWASP	X	OpenSSL	X	Jenkins	N/A
	OpenStack	X	PIWIK	X	Eclipse	N/A
	Spring FW	X	Red Hat	N/A	Bitergia	X
	LibreOffice	?	Drupal	X	OwnCloud	X
	Debian	X	Apache Tomcat	?		

BP42. Best Practice Name:	**Use PHP Snippets Sparingly and with Caution**

Use	**Objectives**	**Benefits**
Site Building	Drupal allows using PHP code in blocks entered through the Back-Office. This offers great power and flexibility, but should be avoided due to performance and security reasons	Ensure site maintenance, performance and security

SDLC Phase Where It Is Used					
Analysis	Design	Development	Testing	Deployment	Maintenance
N/A	N/A	X	N/A	X	X

Roles (i.e. Analyst, Developer, Tester)	Developer, QA
Related Methodologies, Best Practices and Tools	
Related Technologies (i.e. Java)	

FOSS Communities Using This Best Practice	OWASP	?	OpenSSL	N/A	Jenkins	N/A
	OpenStack	N/A	PIWIK	N/A	Eclipse	N/A
	Spring FW	N/A	Red Hat	N/A	Bitergia	N/A
	LibreOffice	?	Drupal	X	OwnCloud	N/A
	Debian	N/A	Apache Tomcat	?		

BP43. Best Practice Name:						**Never Hack the Core**
Use			**Objectives**			**Benefits**
Site Building			Do not modify core files that make up the framework base system			Updates, including security updates, and avoiding issues when applying such updates
SDLC Phase Where It Is Used						
Analysis	Design	Development	Testing	Deployment	Maintenance	
N/A	N/A	X	N/A	N/A	N/A	
Roles (i.e. Analyst, Developer, Tester)						Developer, QA
Related Methodologies, Best Practices and Tools						
Related Technologies (i.e. Java)						

FOSS Communities Using This Best Practice	OWASP	?	OpenSSL	N/A	Jenkins	N/A
	OpenStack	N/A	PIWIK	N/A	Eclipse	N/A
	Spring FW	N/A	Red Hat	N/A	Bitergia	N/A
	LibreOffice	?	Drupal	X	OwnCloud	N/A
	Debian	N/A	Apache Tomcat	?		

BP44. Best Practice Name:	**Avoid Hardcoding**

Use	**Objectives**	**Benefits**
Site Building	Hardcoding is indicative of failure to anticipate factors	Improve the way time-constraint coders use or learn the framework. Understanding of all the ways in which these functions interact

SDLC Phase Where It Is Used					
Analysis	Design	Development	Testing	Deployment	Maintenance
N/A	N/A	X	N/A	N/A	N/A

Roles (i.e. Analyst, Developer, Tester)	Developer, QA
Related Methodologies, Best Practices and Tools	
Related Technologies (i.e. Java)	

FOSS Communities Using This Best Practice	OWASP	X	OpenSSL	N/A	Jenkins	N/A
	OpenStack	X	PIWIK	N/A	Eclipse	N/A
	Spring FW	N/A	Red Hat	N/A	Bitergia	N/A
	LibreOffice	?	Drupal	X	OwnCloud	N/A
	Debian	X	Apache Tomcat	X		

BP45. Best Practice Name:	**Write 'Show All Errors' Compliant Code**

Use	**Objectives**	**Benefits**
Development / Coding Standards	Write code that does not generate redundant warning messages. Set the error reporting level, in non-production servers, to show all errors	Prevents servers from generating large numbers of warning messages in the logs, and helps detect unwanted behaviours of code in special cases

SDLC Phase Where It Is Used					
Analysis	Design	Development	Testing	Deployment	Maintenance
N/A	N/A	X	X	N/A	N/A

Roles (i.e. Analyst, Developer, Tester)	Developer, QA
Related Methodologies, Best Practices and Tools	
Related Technologies (i.e. Java)	

FOSS Communities Using This Best Practice	OWASP	X	OpenSSL	N/A	Jenkins	N/A
	OpenStack	N/A	PIWIK	N/A	Eclipse	N/A
	Spring FW	N/A	Red Hat	N/A	Bitergia	N/A
	LibreOffice	?	Drupal	X	OwnCloud	N/A
	Debian	N/A	Apache Tomcat	?		

BP46. Best Practice Name:	**Disable Unneeded Modules in Production**

Use	**Objectives**	**Benefits**
Configuration / Performance	In Production, disable unneeded modules related to development, debugging, testing and the GUI	Enabling only the modules that are needed reduces system overload and improves performance

SDLC Phase Where It Is Used					
Analysis	Design	Development	Testing	Deployment	Maintenance
N/A	N/A	X	X	X	N/A

Roles (i.e. Analyst, Developer, Tester)	Developer, QA
Related Methodologies, Best Practices and Tools	
Related Technologies (i.e. Java)	

FOSS Communities Using This Best Practice	OWASP	X	OpenSSL	N/A	Jenkins	N/A
	OpenStack	N/A	PIWIK	N/A	Eclipse	N/A
	Spring FW	N/A	Red Hat	X	Bitergia	N/A
	LibreOffice	?	Drupal	X	OwnCloud	N/A
	Debian	X	Apache Tomcat	X		

BP47. Best Practice Name:	**Secure User Accounts**

Use	**Objectives**	**Benefits**
Configuration / Security	Set a strong password for the UID 1 (superadmin) account and change the username if it is a common admin username. Create a separate normal administrator account for normal use. Use Captcha and other modules to secure the login page	Uncommon admin username accounts with strong passwords are harder to crack. By not using the superadmin account helps avoid undesired configuration/management mistakes. However, some actions are only possible with the superadmin account. Brute force protection helps to limit the number of abusive login attempts

SDLC Phase Where It Is Used					
Analysis	Design	Development	Testing	Deployment	Maintenance
N/A	N/A	X	X	X	X

Roles (i.e. Analyst, Developer, Tester)	Developer, Tester, QA
Related Methodologies, Best practices and Tools	
Related Technologies (i.e. Java)	

FOSS Communities Using This Best Practice						
	OWASP	X	OpenSSL	N/A	Jenkins	X
	OpenStack	N/A	PIWIK	N/A	Eclipse	N/A
	Spring FW	N/A	Red Hat	X	Bitergia	N/A
	LibreOffice	?	Drupal	X	OwnCloud	N/A
	Debian	X	Apache Tomcat	X		

BP48. Best Practice Name:	**Secure User Permissions Configuration**

Use	**Objectives**	**Benefits**
Configuration / Security	User accounts created should use e-mail verification and be subject to administrator approval. Control and limit user permissions upon self-registration, content creation and management, file uploads of unsafe file extensions, and site administration	Control of fake user account creation and spam activity on the website. Ensures that critical tasks cannot be performed by untrusted users

SDLC Phase Where It Is Used					
Analysis	Design	Development	Testing	Deployment	Maintenance
N/A	N/A	X	X	X	X

Roles (i.e. Analyst, Developer, Tester)	Developer, Tester, QA
Related Methodologies, Best Practices and Tools	
Related Technologies (i.e. Java)	

FOSS Communities Using This Best Practice	OWASP	?	OpenSSL	N/A	Jenkins	N/A
	OpenStack	N/A	PIWIK	N/A	Eclipse	N/A
	Spring FW	N/A	Red Hat	X	Bitergia	N/A
	LibreOffice	?	Drupal	X	OwnCloud	N/A
	Debian	X	Apache Tomcat	X		

BP49. Best Practice Name:	**Provide Additional Security Through Added Modules**

Use	**Objectives**	**Benefits**
Configuration / Security	Enable any contributed modules that provide security measures and protection	Provide additional protection against vulnerabilities using well-known and community-maintained modules

SDLC Phase Where It Is Used					
Analysis	Design	Development	Testing	Deployment	Maintenance
N/A	N/A	X	X	X	X

Roles (i.e. Analyst, Developer, Tester)	Developer, Tester, QA
Related Methodologies, Best Practices and Tools	
Related Technologies (i.e. Java)	

FOSS Communities Using This Best Practice						
	OWASP	?	OpenSSL	N/A	Jenkins	X
	OpenStack	N/A	PIWIK	N/A	Eclipse	N/A
	Spring FW	N/A	Red Hat	X	Bitergia	N/A
	LibreOffice	?	Drupal	X	OwnCloud	N/A
	Debian	X	Apache Tomcat	?		

BP50. Best Practice Name:	**Apply Core and Contributed Modules Security Updates**

Use	**Objectives**	**Benefits**
Configuration / Security	Install recommended updates as soon as possible. If the security group does not support any modules in pre-release, avoid installing modules in 'dev' and pre-release stages on production sites	The website is protected against the latest known security flaws detected by the community. Prevents using pre-release and non-stable modules that do not provide security updates

SDLC Phase Where It Is Used					
Analysis	Design	Development	Testing	Deployment	Maintenance
N/A	N/A	X	X	X	X

Roles (i.e. Analyst, Developer, Tester)	Developer, Tester, QA
Related Methodologies, Best Practices and Tools	
Related Technologies (i.e. Java)	

FOSS Communities Using This Best Practice	OWASP	X	OpenSSL	N/A	Jenkins	N/A
	OpenStack	N/A	PIWIK	N/A	Eclipse	N/A
	Spring FW	N/A	Red Hat	N/A	Bitergia	N/A
	LibreOffice	?	Drupal	X	OwnCloud	N/A
	Debian	X	Apache Tomcat	?		

BP51. Best Practice Name:	**SEO Best Practices**

Use	**Objectives**	**Benefits**
Configuration / SEO	Enable clean URLs on the site. Define and configure path patterns for the various types of pages (content, taxonomy terms, etc.) Enable SEO characteristics on the site with the help of several contributed modules: Redirect, XML Sitemap, Global Redirect, Google Analytics, and others	Ensure the website responds properly to expectations of end users and search engines. Facilitate to get a good rank in search engines

SDLC Phase Where It Is Used					
Analysis	Design	Development	Testing	Deployment	Maintenance
N/A	N/A	X	X	X	N/A

Roles (i.e. Analyst, Developer, Tester)	Developer, Tester, QA
Related Methodologies, Best Practices and Tools	
Related Technologies (i.e. Java)	

FOSS Communities Using This Best Practice	OWASP	?	OpenSSL	N/A	Jenkins	N/A
	OpenStack	N/A	PIWIK	N/A	Eclipse	N/A
	Spring FW	N/A	Red Hat	N/A	Bitergia	N/A
	LibreOffice	?	Drupal	X	OwnCloud	N/A
	Debian	N/A	Apache Tomcat	?		

BP52. Best Practice Name:			**Critical Software Contributor**			
Use		**Objectives**		**Benefits**		
Known contributors		Only trusted and veteran people can contribute to critical parts of software.		Ensure the quality and security of the code		
SDLC Phase Where It Is Used						
Analysis	Design	Development	Testing	Deployment	Maintenance	
X	X	X	N/A	N/A	NA	
Roles (i.e. Analyst, Developer, Tester)			Analyst, developer			
Related Methodologies, Best Practices and Tools						
Related Technologies (i.e. Java)						

FOSS Communities Using This Best Practice	OWASP	**?**	OpenSSL	X	Jenkins	N/A
	OpenStack	N/A	PIWIK	N/A	Eclipse	N/A
	Spring FW	N/A	Red Hat	N/A	Bitergia	N/A
	LibreOffice	?	Drupal	?	OwnCloud	N/A
	Debian	N/A	Apache Tomcat	?		

BP53. Best Practice Name:			**SecDevOps**			
Use		**Objectives**		**Benefits**		
Integration of the development, security and operation phase in the development team		It ensures that security is carefully implemented from the beginning,		Ensure security from the early phases of the life cycle. Reduce the cost of security implementation during software development		
SDLC Phase Where It Is Used						
Analysis	Design	Development	Testing	Deployment	Maintenance	
N/A	X	X	X	X	X	
Roles (i.e. Analyst, Developer, Tester)			Developer, analyst, project manager			
Related Methodologies, Best Practices and Tools						
Related Technologies (i.e. Java)						
FOSS Communities Using This Best Practice	OWASP	X	OpenSSL	N/A	Jenkins	N/A
	OpenStack	N/A	PIWIK	N/A	Eclipse	N/A
	Spring FW	N/A	Red Hat	N/A	Bitergia	N/A
	LibreOffice	?	Drupal	?	OwnCloud	N/A
	Debian	X	Apache Tomcat	?		

BP54. Best Practice Name:			**Generate LTS (Long Time Support) Releases**			
Use		**Objectives**		**Benefits**		
FOSS community releases a stable version, supported for a longer time than usual.		FOSS version interesting for production environments		Enterprise users will appreciate a stable and supported long version		
SDLC Phase Where It Is Used						
Analysis	Design	Development	Testing	Deployment	Maintenance	
N/A	N/A	N/A	N/A	N/A	X	
Roles (i.e. Analyst, Developer, Tester)			Project Manager			
Related Methodologies, Best Practices and Tools						
Related Technologies (i.e. Java)						

FOSS Communities Using This Best Practice	OWASP	?	OpenSSL	N/A	Jenkins	N/A
	OpenStack	N/A	PIWIK	N/A	Eclipse	X
	Spring FW	N/A	Red Hat	N/A	Bitergia	N/A
	LibreOffice	?	Drupal	?	OwnCloud	N/A
	Debian	N/A	Apache Tomcat	?		

BP55. Best Practice Name:	**Generate LTS (Long Time Support) Releases**

Use	**Objectives**	**Benefits**
Product generate stable version.	Have a stable version of the product in the production	Have a final version of the product and can continue in the developing of alfa and beta versions.

SDLC Phase Where It Is Used					
Analysis	Design	Development	Testing	Deployment	Maintenance
N/A	N/A	N/A	N/A	N/A	X

Roles (i.e. Analyst, Developer, Tester)	
Related Methodologies, Best Practices and Tools	
Related Technologies (i.e. Java)	

FOSS Communities Using This Best Practice	OWASP	?	OpenSSL	N/A	Jenkins	X
	OpenStack	N/A	PIWIK	N/A	Eclipse	X
	Spring FW	N/A	Red Hat	X	Bitergia	N/A
	LibreOffice	?	Drupal	?	OwnCloud	N/A
	Debian	X	Apache Tomcat	?		

3.3. Tools Used by the Analysed FOSS Communities During the Software Development Lifecycle

T1	**Tool Name:**	**GitHub**
Use	**Objectives**	**Benefits**
Web-based Git repository hosting service	Distributed revision control and source code management functionality for Git	Provides a central repository where all developers can push and pull their changes to and from a repository

SDLC Phase Where It Is Used					
Analysis	Design	Development	Testing	Deployment	Maintenance
X	X	N/A	N/A	N/A	N/A

Roles (i.e. Analyst, Developer, Tester)	
Related Methodologies, Best Practices and Tools	N/A
Related Technologies (i.e. Java)	RUBY

FOSS Communities Using This Tool	OWASP	X	OpenSSL	X	Jenkins	X
	OpenStack	X	PIWIK	X	Eclipse	X
	Spring FW	X	Red Hat	N/A	Bitergia	N/A
	LibreOffice	N/A	Drupal	X	OwnCloud	
	Debian	X	Apache Tomcat	X		

T2	**Tool Name:**	**SVN**

Use	**Objectives**	**Benefits**
Code repository	Mainly used to manage versions and branches of the source code	Provides individual commits, quick and flexible updates/commits, and ease of integration

SDLC Phase Where It Is Used					
Analysis	Design	Development	Testing	Deployment	Maintenance
N/A	N/A	X	N/A	N/A	N/A

Roles (i.e. Analyst, Developer, Tester)	Developer, Tester
Related Methodologies, Best Practices and Tools	Source code versioning
Related Technologies (i.e. Java)	

FOSS Communities Using This Tool	OWASP	N/A	OpenSSL	N/A	Jenkins	X
	OpenStack	N/A	PIWIK	N/A	Eclipse	N/A
	Spring FW	N/A	Red Hat	X	Bitergia	N/A
	LibreOffice	N/A	Drupal	X	OwnCloud	N/A
	Debian	N/A	Apache Tomcat	X		

T3	**Tool Name:**	**Jenkins**

Use	**Objectives**	**Benefits**
Continuous integration	Automates builds, tests, and releases	Deployment automation. Test results reporting

SDLC Phase Where It Is Used					
Analysis	Design	Development	Testing	Deployment	Maintenance
X	X	N/A	N/A	N/A	X

Roles (i.e. Analyst, Developer, Tester)	Developer, Tester, IT Operations
Related Methodologies, Best Practices and Tools	Continuous integration, Cruise control, Bamboo, Nexus, SVN, GitHub
Related Technologies (i.e. Java)	JAVA

FOSS Communities Using This Tool	OWASP	N/A	OpenSSL	N/A	Jenkins	X
	OpenStack	X	PIWIK	N/A	Eclipse	N/A
	Spring FW	N/A	Red Hat	X	Bitergia	N/A
	LibreOffice	N/A	Drupal	X	OwnCloud	N/A
	Debian	N/A	Apache Tomcat	X		

T4	Tool Name:	OWASP ESAPI

Use	Objectives	Benefits
This API is designed to automatically handle many aspects of application security	Develop a secure web application. Use an interface with its own implementation based on other infrastructures. Use the interfaces with the reference implementation as a starting point	Cost savings from reduced development timeframes. Increased security from the use of strongly analysed and carefully designed methods

SDLC Phase Where It Is Used					
Analysis	Design	Development	Testing	Deployment	Maintenance
N/A	N/A	X	N/A	N/A	N/A

Roles (i.e. Analyst, Developer, Tester)	Developer
Related Methodologies, Best Practices and Tools	
Related Technologies (i.e. Java)	Java, .NET, PHP

FOSS Communities Using This Tool	OWASP	X	OpenSSL	N/A	Jenkins	N/A
	OpenStack	N/A	PIWIK	N/A	Eclipse	N/A
	Spring FW	N/A	Red Hat	N/A	Bitergia	N/A
	LibreOffice	N/A	Drupal	N/A	OwnCloud	N/A
	Debian	N/A	Apache Tomcat	N/A		

T5	**Tool Name:**	**OWASP Zed Attack Proxy**

Use	**Objectives**	**Benefits**
Integrated penetration testing tool used to find security vulnerabilities in web applications	Help users develop and apply application security skills. Find web application vulnerabilities. Automated scanners	Designed to be easy to use. Provides an extensible platform for testing. Raises the bar for other security tools

SDLC Phase Where It Is Used					
Analysis	Design	Development	Testing	Deployment	Maintenance
N/A	N/A	X	X	N/A	N/A

Roles (i.e. Analyst, Developer, Tester)	Developer, Tester
Related Methodologies, Best Practices and Tools	
Related Technologies (i.e. Java)	Java

FOSS Communities Using This Tool	OWASP	X	OpenSSL	N/A	Jenkins	N/A
	OpenStack	N/A	PIWIK	N/A	Eclipse	N/A
	Spring FW	N/A	Red Hat	N/A	Bitergia	N/A
	LibreOffice	N/A	Drupal	N/A	OwnCloud	N/A
	Debian	N/A	Apache Tomcat	N/A		

T6	Tool Name:	OWASP AppSensor

Use	Objectives	Benefits
This tool is used to implement intrusion detection and automated response into applications	Build a robust system for attack detection. System analysis. Response within an enterprise application	AppSensor defines over 50 different detection points which can be used to identify a malicious attacker. This tool provides guidance on how to respond once a malicious attacker has been identified. It is possible to identify and eliminate an attack threat before it is able to successfully identify an exploitable flaw

SDLC Phase Where It Is Used					
Analysis	Design	Development	Testing	Deployment	Maintenance
N/A	N/A	N/A	X	X	X

Roles (i.e. Analyst, Developer, Tester)	Architects, Developers, Security Analysts and System Administrators
Related Methodologies, Best Practices and Tools	
Related Technologies (i.e. Java)	

FOSS Communities Using This Tool	OWASP	X	OpenSSL	N/A	Jenkins	N/A
	OpenStack	N/A	PIWIK	N/A	Eclipse	N/A
	Spring FW	N/A	Red Hat	N/A	Bitergia	N/A
	LibreOffice	N/A	Drupal	N/A	OwnCloud	N/A
	Debian	N/A	Apache Tomcat	N/A		

T7	Tool Name:	JUNIT

Use	Objectives	Benefits
A unit testing provider for the Java programming language. It is used for unit testing of Java applications	Running Java classes in a controlled way. Evaluating whether the running class methods behave as expected. Control regression testing	Simple to use. Allows testing a single class at a time or a suite of tests for a group of classes. Increases confidence in the correctness of your code. Improves the tested class design. Failure of a test is clearly evident

SDLC Phase Where It Is Used					
Analysis	Design	Development	Testing	Deployment	Maintenance
N/A	N/A	N/A	X	N/A	N/A

Roles (i.e. Analyst, Developer, Tester)	
Related Methodologies, Best Practices and Tools	NetBeans, Eclipse (plugins)
Related Technologies (i.e. Java)	Java

FOSS Communities Using This Tool	OWASP	N/A	OpenSSL	N/A	Jenkins	X
	OpenStack	N/A	PIWIK	N/A	Eclipse	N/A
	Spring FW	X	Red Hat	N/A	Bitergia	N/A
	LibreOffice	N/A	Drupal	N/A	OwnCloud	N/A
	Debian	X	Apache Tomcat	X		

T8	Tool Name:	Apache Gump

Use	Objectives	Benefits
This Apache software aims to build and test all open source Java projects	Make sure all projects are compatible at the API level, and in terms of matching functionality specifications	Detects incompatible changes to the software in just hours after they have been checked into the version control system. Notifications are sent to the project team the moment such a change is detected, referencing additional details that are available via reports online

SDLC Phase Where It Is Used					
Analysis	Design	Development	Testing	Deployment	Maintenance
N/A	N/A	X	X	N/A	N/A

Roles (i.e. Analyst, Developer, Tester)	Developer, Tester
Related Methodologies, Best Practices and Tools	Apache Ant, Apache Maven
Related Technologies (i.e. Java)	Python

FOSS Communities Using This Tool	OWASP	N/A	OpenSSL	N/A	Jenkins	N/A
	OpenStack	N/A	PIWIK	N/A	Eclipse	N/A
	Spring FW	N/A	Red Hat	N/A	Bitergia	N/A
	LibreOffice	N/A	Drupal	N/A	OwnCloud	N/A
	Debian	N/A	Apache Tomcat	X		

T9	Tool Name:	Apache Ant

Use	Objectives	Benefits
Apache Ant is a software tool for automating software build processes	Compile, assemble, test and run Java applications. Build non-Java applications, i.e. C or C++ applications. Pilot any type of process that can be described in terms of targets and tasks.	Ant is extremely flexible and does not impose coding conventions or directory layouts to the Java projects which adopt it as a build tool. Users of Ant can develop their own 'antlibs' that contain Ant tasks and types; a large number of such ready-made commercial or open-source 'antlibs' are currently available.

SDLC Phase Where It Is Used					
Analysis	Design	Development	Testing	Deployment	Maintenance
N/A	N/A	X	N/A	N/A	N/A

Roles (i.e. Analyst, Developer, Tester)	
Related Methodologies, Best Practices and Tools	
Related Technologies (i.e. Java)	Java, XML

FOSS Communities Using This Tool	OWASP	N/A	OpenSSL	N/A	Jenkins	N/A
	OpenStack	N/A	PIWIK	N/A	Eclipse	N/A
	Spring FW	N/A	Red Hat	N/A	Bitergia	N/A
	LibreOffice	N/A	Drupal	N/A	OwnCloud	N/A
	Debian	X	Apache Tomcat	X		

T10	Tool Name:	OpenStack NOVA

Use	Objectives	Benefits
The Nova, OpenStack Compute service is used for hosting and managing cloud computing systems	Nova is built on a messaging architecture and all of its components can typically be run on several servers. This architecture allows the components to communicate through a message queue. Deferred objects are used to avoid blocking while a component waits in the message queue for a response	It provides massive, scalable on-demand self-service access to computing resources

SDLC Phase Where It Is Used					
Analysis	Design	Development	Testing	Deployment	Maintenance
N/A	N/A	X	N/A	N/A	N/A

Roles (i.e. Analyst, Developer, Tester)	Developer
Related Methodologies, Best Practices and Tools	GitHub
Related Technologies (i.e. Java)	Python

FOSS Communities Using This Tool	OWASP	N/A	OpenSSL	N/A	Jenkins	N/A
	OpenStack	X	PIWIK	N/A	Eclipse	N/A
	Spring FW	N/A	Red Hat	N/A	Bitergia	N/A
	LibreOffice	N/A	Drupal	N/A	OwnCloud	N/A
	Debian	N/A	Apache Tomcat	N/A		

T11	**Tool Name:**	**OpenStack Zuul**

Use	**Objectives**	**Benefits**
Continuous integration	Provide a gateway between Gerrit and Jenkins	Merges a change in the repository only when it passes a pre-defined test

SDLC Phase Where It Is Used					
Analysis	Design	Development	Testing	Deployment	Maintenance
N/A	N/A	X	X	N/A	N/A

Roles (i.e. Analyst, Developer, Tester)	Developer
Related Methodologies, Best Practices and Tools	GitHub, Gerrit, Jenkins.
Related Technologies (i.e. Java)	Python

FOSS Communities Using This Tool	OWASP	N/A	OpenSSL	N/A	Jenkins	N/A
	OpenStack	X	PIWIK	N/A	Eclipse	N/A
	Spring FW	N/A	Red Hat	N/A	Bitergia	N/A
	LibreOffice	N/A	Drupal	N/A	OwnCloud	N/A
	Debian	N/A	Apache Tomcat	N/A		

T12	**Tool Name:**	**OpenStack Bandit**

Use	**Objectives**	**Benefits**
Security linter for Python source code	To convert source code into a parsed tree of Python syntax node	Maintains the security in OpenStack projects when it is used as a gate test

SDLC Phase Where It Is Used					
Analysis	Design	Development	Testing	Deployment	Maintenance
N/A	N/A	X	X	N/A	N/A

Roles (i.e. Analyst, Developer, Tester)	Developer, Tester
Related Methodologies, Best Practices and Tools	
Related Technologies (i.e. Java)	Python

FOSS Communities Using This Tool	OWASP	N/A	OpenSSL	N/A	Jenkins	N/A
	OpenStack	X	PIWIK	N/A	Eclipse	N/A
	Spring FW	N/A	Red Hat	N/A	Bitergia	N/A
	LibreOffice	N/A	Drupal	N/A	OwnCloud	N/A
	Debian	N/A	Apache Tomcat	N/A		

T13	**Tool Name:**	**Composer**

Use	**Objectives**	**Benefits**
Dependency management in PHP	Allow declaration and management of the libraries a project depends on (including installation/updating)	Enables declaration of the dependant libraries. Offers dependency version checking of packages; downloads and installs them if deemed necessary

SDLC Phase Where It Is Used					
Analysis	Design	Development	Testing	Deployment	Maintenance
N/A	N/A	X	N/A	N/A	X

Roles (i.e. Analyst, Developer, Tester)	Developer
Related Methodologies, Best Practices and Tools	GitHub
Related Technologies (i.e. Java)	PHP

FOSS Communities Using This Tool	OWASP	N/A	OpenSSL	N/A	Jenkins	N/A
	OpenStack	N/A	PIWIK	X	Eclipse	N/A
	Spring FW	N/A	Red Hat	N/A	Bitergia	N/A
	LibreOffice	N/A	Drupal	N/A	OwnCloud	N/A
	Debian	N/A	Apache Tomcat	N/A		

T14	Tool Name:	TortoiseSVN

Use	Objectives	Benefits
Revision control, version control and source control software	Allow declaration and management of the libraries a project depends on (including installation/updating)	Enables declaration of the dependant libraries. Offers dependency version checking of packages; downloads and installs them if deemed necessary

SDLC Phase Where It Is Used					
Analysis	Design	Development	Testing	Deployment	Maintenance
N/A	N/A	X	X	N/A	N/A

Roles (i.e. Analyst, Developer, Tester)	Developer
Related Methodologies, Best Practices and Tools	Apache SVN
Related Technologies (i.e. Java)	

FOSS Communities Using This Tool	OWASP	N/A	OpenSSL	N/A	Jenkins	N/A
	OpenStack	N/A	PIWIK	N/A	Eclipse	N/A
	Spring FW	N/A	Red Hat	N/A	Bitergia	N/A
	LibreOffice	N/A	Drupal	N/A	OwnCloud	N/A
	Debian	N/A	Apache Tomcat	X		

T15	**Tool Name:**	**OpenHUB**

Use	**Objectives**	**Benefits**
Revision control repository	Offer analytics and search services for discovering, evaluating, tracking, and comparing open source code and projects	It is editable by everyone, like a wiki. It provides reports about the composition and activity of project code bases and aggregates the data to track the changing demographics of the FOSS world

SDLC Phase Where It Is Used					
Analysis	Design	Development	Testing	Deployment	Maintenance
X	X	X	X	N/A	X

Roles (i.e. Analyst, Developer, Tester)	Tester, Developer
Related Methodologies, Best Practices and Tools	N/A
Related Technologies (i.e. Java)	RUBY

FOSS Communities Using This Tool	OWASP	N/A	OpenSSL	N/A	Jenkins	N/A
	OpenStack	X	PIWIK	N/A	Eclipse	N/A
	Spring FW	N/A	Red Hat	N/A	Bitergia	N/A
	LibreOffice	N/A	Drupal	N/A	OwnCloud	N/A
	Debian	N/A	Apache Tomcat	X		

T16	Tool Name:	Bugzilla

Use	Objectives	Benefits
Bugzilla is a web-based general-purpose bug tracker and testing tool	Organize software defects by utilising different means. Categorize software defects according to their priority and severity, and assign versions for resolution	Allows monitoring of multiple products with different versions. Allows commenting, assigning of issues, tracking of proposed solutions and their fixes of problematic components

SDLC Phase Where It Is Used					
Analysis	Design	Development	Testing	Deployment	Maintenance
N/A	N/A	N/A	X	X	X

Roles (i.e. . Analyst, Developer, Tester)	Developer, Tester
Related Methodologies, Best Practices and Tools	Apache (Server), MySQL (database)
Related Technologies (i.e. Java)	Perl

FOSS Communities Using This Tool	OWASP	N/A	OpenSSL	N/A	Jenkins	X
	OpenStack	N/A	PIWIK	N/A	Eclipse	X
	Spring FW	N/A	Red Hat	X	Bitergia	N/A
	LibreOffice	N/A	Drupal	N/A	OwnCloud	N/A
	Debian	X	Apache Tomcat	X		

T17	**Tool Name:**	**OpenStack Anchor**

Use	**Objectives**	**Benefits**
Ephemeral PKI (Public Key Infrastructure) system	Enable cryptographic trust in OpenStack services in a way that does not rely on broken provisioning and revocation mechanisms that undermine most PKI deployments	Trustworthiness in cryptographic services

SDLC Phase Where It Is Used					
Analysis	Design	Development	Testing	Deployment	Maintenance
N/A	N/A	X	X	N/A	N/A

Roles (i.e. . Analyst, Developer, Tester)	Developer, Tester
Related Methodologies, Best Practices and Tools	OpenStack Gerrit
Related Technologies (i.e. Java)	

FOSS Communities Using This Tool	OWASP	N/A	OpenSSL	N/A	Jenkins	N/A
	OpenStack	X	PIWIK	N/A	Eclipse	N/A
	Spring FW	N/A	Red Hat	N/A	Bitergia	N/A
	LibreOffice	N/A	Drupal	N/A	OwnCloud	N/A
	Debian	N/A	Apache Tomcat	N/A		

T18	**Tool Name:**	**Launchpad**

Use	**Objectives**	**Benefits**
Website and open source web application that supports software development	Community support site and Knowledge Base. A system for tracking specifications and new features. Bug tracking. Source code hosting	Having an efficient platform to support FOSS development and maintenance

SDLC Phase Where It Is Used					
Analysis	Design	Development	Testing	Deployment	Maintenance
N/A	N/A	X	X	X	X

Roles (i.e. . Analyst, Developer, Tester)	Developer, Tester
Related Methodologies, Best Practices and Tools	
Related Technologies (i.e. Java)	

FOSS Communities Using This Tool	OWASP	N/A	OpenSSL	N/A	Jenkins	N/A
	OpenStack	X	PIWIK	N/A	Eclipse	N/A
	Spring FW	N/A	Red Hat	N/A	Bitergia	N/A
	LibreOffice	X	Drupal	N/A	OwnCloud	N/A
	Debian	N/A	Apache Tomcat	N/A		

T19	**Tool Name:**	**Gerrit**

Use	**Objectives**	**Benefits**
Web-based team code collaboration tool	Gerrit is intended to provide a lightweight framework for reviewing every commit before it is accepted into the code base	Web application for managing code contribution and code review

SDLC Phase Where It Is Used					
Analysis	Design	Development	Testing	Deployment	Maintenance
N/A	N/A	X	X	N/A	N/A

Roles (i.e. . Analyst, Developer, Tester)	Developer, Tester
Related Methodologies, Best Practices and Tools	
Related Technologies (i.e. Java)	Git

FOSS Communities Using This Tool	OWASP	N/A	OpenSSL	N/A	Jenkins	N/A
	OpenStack	X	PIWIK	N/A	Eclipse	X
	Spring FW	N/A	Red Hat	N/A	Bitergia	N/A
	LibreOffice	X	Drupal	N/A	OwnCloud	N/A
	Debian	N/A	Apache Tomcat	N/A		

T20	**Tool Name:**	**Jira**

Use	**Objectives**	**Benefits**
Bug tracking system Project management software	It provides bug tracking, issue tracking, and project management functions.	Connection to all the developer tools that it uses, making it the single source of truth for every step in their projects.

SDLC Phase Where It Is Used					
Analysis	Design	Development	Testing	Deployment	Maintenance
X	X	X	X	X	X

Roles (i.e. . Analyst, Developer, Tester)	IT Team
Related Methodologies, Best Practices and Tools	PM2 and development methodology
Related Technologies (i.e. Java)	Java

FOSS Communities Using This Tool	OWASP	N/A	OpenSSL	N/A	Jenkins	X
	OpenStack	N/A	PIWIK	N/A	Eclipse	N/A
	Spring FW	X	Red Hat	N/A	Bitergia	N/A
	LibreOffice	N/A	Drupal	N/A	OwnCloud	N/A
	Debian	N/A	Apache Tomcat	N/A		

T21	**Tool Name:**	**Apache Solr**

Use	**Objectives**	**Benefits**
Enterprise search server with a REST-like API.	Text search engine for enterprise environments.	A popular, blazing-fast, open source enterprise search platform.

SDLC Phase Where It Is Used					
Analysis	Design	Development	Testing	Deployment	Maintenance
N/A	X	N/A	N/A	N/A	N/A

Roles (i.e. . Analyst, Developer, Tester)	Analyst
Related Methodologies, Best Practices and Tools	
Related Technologies (i.e. Java)	Java, XML/HTTP, JSON, PHP, RUBY, Python

FOSS Communities Using This Tool	OWASP	N/A	OpenSSL	N/A	Jenkins	N/A
	OpenStack	N/A	PIWIK	N/A	Eclipse	N/A
	Spring FW	N/A	Red Hat	N/A	Bitergia	N/A
	LibreOffice	N/A	Drupal	X	OwnCloud	N/A
	Debian	N/A	Apache Tomcat	N/A		

T22	**Tool Name:**	**OpenVas**

Use	**Objectives**	**Benefits**
It is oriented to vulnerability scanning and vulnerability management solution	Current status of web application vulnerabilities.	Knowledge about possible website vulnerabilities. It improves security in websites by detecting possible flaws.

SDLC Phase Where It Is Used					
Analysis	Design	Development	Analysis	Design	Development
N/A	N/A	N/A	N/A	N/A	N/A

Roles (i.e. . Analyst, Developer, Tester)	Security Auditor
Related Methodologies, Best Practices and Tools	
Related Technologies (i.e. Java)	SSL

FOSS Communities Using This Tool	OWASP	N/A	OpenSSL	N/A	Jenkins	N/A
	OpenStack	N/A	PIWIK	N/A	Eclipse	N/A
	Spring FW	N/A	Red Hat	X	Bitergia	N/A
	LibreOffice	N/A	Drupal	X	OwnCloud	N/A
	Debian	N/A	Apache Tomcat	N/A		

T23	**Tool Name:**	**Coverity**

Use	**Objectives**	**Benefits**
Software testing and static analysis tool	Find defects and security vulnerabilities in source code	Fix critical defects quickly and efficient. Reduce the risk of costly and brand-damaging software failures and security breaches in the field or in production

SDLC Phase Where It Is Used					
Analysis	Design	Development	Testing	Deployment	Maintenance
X	N/A	N/A	X	X	X

Roles (i.e. . Analyst, Developer, Tester)	Analyst, tester.
Related Methodologies, Best Practices and Tools	
Related Technologies (i.e. Java)	C, C++, Java

FOSS Communities Using This Tool	OWASP	N/A	OpenSSL	N/A	Jenkins	N/A
	OpenStack	N/A	PIWIK	N/A	Eclipse	N/A
	Spring FW	N/A	Red Hat	N/A	Bitergia	N/A
	LibreOffice	X	Drupal	N/A	OwnCloud	N/A
	Debian	N/A	Apache Tomcat	N/A		

T24	**Tool Name:**	**FusionForge(Alioth)**

Use	**Objectives**	**Benefits**
Create and control access to SCM (Software Configuration Management) repositories.	Increase the team collaboration with a several number of tools	Manage file releases, surveys for users and admin, issue tracking with unlimited number of categories.

SDLC Phase Where It Is Used					
Analysis	Design	Development	Testing	Deployment	Maintenance
X	X	X	X	X	X

Roles (i.e. . Analyst, Developer, Tester)	Analyst, developer, tester
Related Methodologies, Best Practices and Tools	
Related Technologies (i.e. Java)	

FOSS Communities Using This Tool	OWASP	N/A	OpenSSL	N/A	Jenkins	N/A
	OpenStack	N/A	PIWIK	N/A	Eclipse	N/A
	Spring FW	N/A	Red Hat	N/A	Bitergia	N/A
	LibreOffice	N/A	Drupal	N/A	OwnCloud	N/A
	Debian	X	Apache Tomcat	N/A		

T25	**Tool Name:**	**Jabber**

Use	**Objectives**	**Benefits**
Extensible messaging and presence protocol	Provides a communications protocol for message-oriented middleware.	It enables the near-real-time of structured yet extensible data between any two or more network entities.

SDLC Phase Where It Is Used					
Analysis	Design	Development	Testing	Deployment	Maintenance
N/A	N/A	N/A	N/A	N/A	X

Roles (i.e. . Analyst, Developer, Tester)	Project Team
Related Methodologies, Best Practices and Tools	
Related Technologies (i.e. Java)	XML

FOSS Communities Using This Tool	OWASP	N/A	OpenSSL	N/A	Jenkins	N/A
	OpenStack	N/A	PIWIK	N/A	Eclipse	N/A
	Spring FW	N/A	Red Hat	N/A	Bitergia	N/A
	LibreOffice	N/A	Drupal	N/A	OwnCloud	N/A
	Debian	X	Apache Tomcat	N/A		

T26	**Tool Name:**	**Rats**

Use	**Objectives**	**Benefits**
Code Review	It allows for finding potential risk problems in several programming languages	Increase security in a source code files

SDLC Phase Where It Is Used					
Analysis	Design	Development	Testing	Deployment	Maintenance
X	N/A	N/A	X	X	X

Roles (i.e. . Analyst, Developer, Tester)	Analyst, tester
Related Methodologies, Best Practices and Tools	
Related Technologies (i.e. Java)	C, C++, Perl, PHP, Python

FOSS Communities Using This Tool	OWASP	N/A	OpenSSL	N/A	Jenkins	N/A
	OpenStack	N/A	PIWIK	N/A	Eclipse	N/A
	Spring FW	N/A	Red Hat	N/A	Bitergia	N/A
	LibreOffice	N/A	Drupal	N/A	OwnCloud	N/A
	Debian	X	Apache Tomcat	N/A		

T27	**Tool Name:**	**FlawFinder**

Use	**Objectives**	**Benefits**
Code review	Find some security holes in the area of buffer overflow . Examines the source code.	Mark the functions in its basic function considered dangerous.

SDLC Phase Where It Is Used					
Analysis	Design	Development	Testing	Deployment	Maintenance
N/A	N/A	N/A	X	X	X

Roles (i.e. . Analyst, Developer, Tester)	Tester
Related Methodologies, Best Practices and Tools	
Related Technologies (i.e. Java)	C, C++

FOSS Communities Using This Tool	OWASP	N/A	OpenSSL	N/A	Jenkins	N/A
	OpenStack	N/A	PIWIK	N/A	Eclipse	N/A
	Spring FW	N/A	Red Hat	N/A	Bitergia	N/A
	LibreOffice	N/A	Drupal	N/A	OwnCloud	N/A
	Debian	X	Apache Tomcat	N/A		

T28	**Tool Name:**	**Pscan**

Use	**Objectives**	**Benefits**
Code review	It is focused on potential threat detection	Increase the detection rate of potential threat and provide more security to the application

SDLC Phase Where It Is Used					
Analysis	Design	Development	Testing	Deployment	Maintenance
N/A	N/A	N/A	X	X	X

Roles (i.e. . Analyst, Developer, Tester)	Tester
Related Methodologies, Best Practices and Tools	
Related Technologies (i.e. Java)	

FOSS Communities Using This Tool	OWASP	N/A	OpenSSL	N/A	Jenkins	N/A
	OpenStack	N/A	PIWIK	N/A	Eclipse	N/A
	Spring FW	N/A	Red Hat	N/A	Bitergia	N/A
	LibreOffice	N/A	Drupal	N/A	OwnCloud	N/A
	Debian	X	Apache Tomcat	N/A		

T29	**Tool Name:**	**Equinox Security**

Use	**Objectives**	**Benefits**
Authentication and authorization functionality.	It provides security functionality to protect their data by means of access control and authentication mechanisms.	It provide a user authentication framework, mechanism for code authorization to protect against potentially malicious code packaged and distributed as bundles.

SDLC Phase Where It Is Used					
Analysis	Design	Development	Analysis	Design	Development
N/A	X	X	N/A	X	X

Roles (i.e. . Analyst, Developer, Tester)	Analyst, Architect
Related Methodologies, Best Practices and Tools	
Related Technologies (i.e. Java)	Java

FOSS Communities Using This Tool	OWASP	N/A	OpenSSL	N/A	Jenkins	N/A
	OpenStack	N/A	PIWIK	N/A	Eclipse	X
	Spring FW	N/A	Red Hat	N/A	Bitergia	N/A
	LibreOffice	N/A	Drupal	N/A	OwnCloud	N/A
	Debian	N/A	Apache Tomcat	N/A		

T30	**Tool Name:**	**Eclipse**

Use	**Objectives**	**Benefits**
Integrated development environment (IDE) and programming tool	It is focused on developing Java applications, but it supports other programming languages such as C. C++, PHP, Python…	Capacity to programming in several language programming languages and architecture.

SDLC Phase Where It Is Used					
Analysis	Design	Development	Analysis	Design	Development
X	X	X	X	X	X

Roles (i.e. . Analyst, Developer, Tester)	Analyst, developer, tester.
Related Methodologies, Best Practices and Tools	
Related Technologies (i.e. Java)	Java

FOSS Communities Using This Tool	OWASP	N/A	OpenSSL	N/A	Jenkins	N/A
	OpenStack	N/A	PIWIK	N/A	Eclipse	X
	Spring FW	X	Red Hat	N/A	Bitergia	N/A
	LibreOffice	N/A	Drupal	N/A	OwnCloud	N/A
	Debian	N/A	Apache Tomcat	N/A		

T31	**Tool Name:**	**Confluence**

Use	**Objectives**	**Benefits**
Team collaboration software	Organize work, create documents, and discussion board in one place	It has been adapted to work with Jira and other Atlassian Software such as Bamboo.

SDLC Phase Where It Is Used					
Analysis	Design	Development	Analysis	Design	Development
X	X	X	X	X	X

Roles (i.e. . Analyst, Developer, Tester)	Project Team
Related Methodologies, Best Practices and Tools	JIRA, BAMBOO
Related Technologies (i.e. Java)	Java

FOSS Communities Using This Tool	OWASP	N/A	OpenSSL	N/A	Jenkins	X
	OpenStack	N/A	PIWIK	N/A	Eclipse	N/A
	Spring FW	N/A	Red Hat	N/A	Bitergia	N/A
	LibreOffice	N/A	Drupal	N/A	OwnCloud	N/A
	Debian	N/A	Apache Tomcat	N/A		

T32	**Tool Name:**	**FishEye**

Use	**Objectives**	**Benefits**
Revision-control browser	Compare different revision of folder, branches or tags.	It provides monitoring and user-level notifications via e-mail or RSS

SDLC Phase Where It Is Used					
Analysis	Design	Development	Analysis	Design	Development
N/A	N/A	X	N/A	N/A	X

Roles (i.e. . Analyst, Developer, Tester)	Developer, Tester
Related Methodologies, Best Practices and Tools	
Related Technologies (i.e. Java)	Java

FOSS Communities Using This Tool	OWASP	N/A	OpenSSL	N/A	Jenkins	X
	OpenStack	N/A	PIWIK	N/A	Eclipse	N/A
	Spring FW	N/A	Red Hat	N/A	Bitergia	N/A
	LibreOffice	N/A	Drupal	N/A	OwnCloud	N/A
	Debian	N/A	Apache Tomcat	N/A		

T33	**Tool Name:**	**MozTrap**

Use	**Objectives**	**Benefits**
Test Case management system.	It allows a several web-testing with the use of different locales and operating systems.	Record and reported the results of test with simple steps.

SDLC Phase Where It Is Used					
Analysis	Design	Development	Testing	Deployment	Maintenance
N/A	N/A	N/A	X	X	X

Roles (i.e. . Analyst, Developer, Tester)	Tester
Related Methodologies, Best Practices and Tools	
Related Technologies (i.e. Java)	

FOSS Communities Using This Tool	OWASP	N/A	OpenSSL	N/A	Jenkins	N/A
	OpenStack	N/A	PIWIK	N/A	Eclipse	N/A
	Spring FW	N/A	Red Hat	N/A	Bitergia	N/A
	LibreOffice	X	Drupal	N/A	OwnCloud	N/A
	Debian	N/A	Apache Tomcat	N/A		

T34	**Tool Name:**	**CMake**

Use	**Objectives**	**Benefits**
Cross-platform for generation and automation code.	Provides a several tools for construction, testing and package of software	It handle in-place and out-place builds, enabling several builds from the same source tree and cross-compilation.

SDLC Phase Where It Is Used					
Analysis	Design	Development	Testing	Deployment	Maintenance
X	X	X	X	N/A	N/A

Roles (i.e. . Analyst, Developer, Tester)	Developer.
Related Methodologies, Best Practices and Tools	
Related Technologies (i.e. Java)	C, C++

FOSS Communities Using This Tool						
	OWASP	N/A	OpenSSL	N/A	Jenkins	N/A
	OpenStack	N/A	PIWIK	N/A	Eclipse	N/A
	Spring FW	N/A	Red Hat	N/A	Bitergia	N/A
	LibreOffice	N/A	Drupal	N/A	OwnCloud	X
	Debian	N/A	Apache Tomcat	N/A		

T35 Tool Name:			**SonarQube**		
Use			**Objectives**		**Benefits**
Continuous code quality and security management			Improve software security and quality to increase the efficiency of development teams and longevity of applications.		Offers reports on duplicated code, code standards, unit test, code complexity, potential bugs, comments and design and architecture.
SDLC Phase Where It Is Used					
Analysis	Design	Development	Analysis	Design	Development
N/A	N/A	X	N/A	N/A	X
Roles (i.e. . Analyst, Developer, Tester)			Tester, Security Auditor, Developer		
Related Methodologies, Best Practices and Tools			Eclipse, Apache Ant.		
Related Technologies (i.e. Java)			Java, Ruby, PHP, .NET		
FOSS Communities Using This Tool	OWASP	N/A	OpenSSL	N/A	Jenkins	N/A
	OpenStack	N/A	PIWIK	N/A	Eclipse	N/A
	Spring FW	X	Red Hat	N/A	Bitergia	N/A
	LibreOffice	N/A	Drupal	N/A	OwnCloud	N/A
	Debian	N/A	Apache Tomcat	N/A		

T36	**Tool Name:**	**IRC**

Use	**Objectives**	**Benefits**
Application layer protocol	It facilitates communication in the form of text. It works on a client/server networking model	Increase the communications between chat servers to other clients. It increases one-on-one communication with private messages

SDLC Phase Where It Is Used					
Analysis	Design	Development	Testing	Deployment	Maintenance
N/A	N/A	N/A	N/A	X	N/A

Roles (i.e. . Analyst, Developer, Tester)	Project Team
Related Methodologies, Best Practices and Tools	
Related Technologies (i.e. Java)	

FOSS Communities Using This Tool	OWASP	X	OpenSSL	X	Jenkins	X
	OpenStack	X	PIWIK	X	Eclipse	X
	Spring FW	X	Red Hat	N/A	Bitergia	X
	LibreOffice	X	Drupal	X	OwnCloud	X
	Debian	X	Apache Tomcat	X		

3.4. Libraries and Building Blocks Used by the Analysed FOSS Communities During the Software Development Lifecycle

LB&B1	**Library Name:**	**OpenSSL**

Use	**Objectives**	**Benefits**
Security library	Software library to be used in applications that need to secure communications against eavesdropping or need to ascertain the identity of the party at the other end	It implements basic cryptographic functions

SDLC Phase Where It Is Used					
Analysis	Design	Development	Testing	Deployment	Maintenance
N/A	N/A	N/A	N/A	N/A	N/A

Roles (i.e. Analyst, Developer, Tester)	
Related Methodologies, Best Practices and Tools	Encryption
Related Technologies (i.e. Java)	

FOSS Communities Using This Library	OWASP	N/A	OpenSSL	X	Jenkins	N/A
	OpenStack	N/A	PIWIK	N/A	Eclipse	N/A
	Spring FW	N/A	Red Hat	N/A	Bitergia	N/A
	LibreOffice	N/A	Drupal	N/A	OwnCloud	N/A
	Debian	N/A	Apache Tomcat	X		

LB&B2	Tool Name:	MySQL

Use	Objectives	Benefits
Multiuser and multithreaded Relational Database Management System. Comes in proprietary and open source versions	Keep data and its relations	Effective persistence module based on SQL technology

SDLC Phase Where It Is Used					
Analysis	Design	Development	Testing	Deployment	Maintenance
N/A	X	X	X	X	X

Roles (i.e. . Analyst, Developer, Tester)	Analyst, Architect, Developer, Tester, System Administrator
Related Methodologies, Best Practices and Tools	
Related Technologies (i.e. Java)	SQL

FOSS Communities Using This Library	OWASP	N/A	OpenSSL	N/A	Jenkins	N/A
	OpenStack	N/A	PIWIK	X	Eclipse	N/A
	Spring FW	N/A	Red Hat	N/A	Bitergia	N/A
	LibreOffice	N/A	Drupal	N/A	OwnCloud	N/A
	Debian	N/A	Apache Tomcat	N/A		

LB&B3	**Tool Name:**	**OpenSSH**

Use	**Objectives**	**Benefits**
Suite of security-related network-level utilities	It helps to secure network communications	Encryption of network traffic over multiple authentication methods and providing secure tunnelling capabilities

SDLC Phase Where It Is Used					
Analysis	Design	Development	Testing	Deployment	Maintenance
N/A	X	X	X	X	X

Roles (i.e. . Analyst, Developer, Tester)	Architect, Analyst, Developer.
Related Methodologies, Best Practices and Tools	Encryption
Related Technologies (i.e. Java)	SSH

FOSS Communities Using This Library	OWASP	N/A	OpenSSL	N/A	Jenkins	N/A
	OpenStack	N/A	PIWIK	N/A	Eclipse	N/A
	Spring FW	N/A	Red Hat	N/A	Bitergia	N/A
	LibreOffice	N/A	Drupal	N/A	OwnCloud	N/A
	Debian	X	Apache Tomcat	N/A		

LB&B4	Tool Name:	GnuPG

Use	Objectives	Benefits
Hybrid-encryption software program	Exchange of secure key	It allows a cryptographic digital signature to a message to verified sender and integrity

SDLC Phase Where It Is Used					
Analysis	Design	Development	Testing	Deployment	Maintenance
N/A	X	X	X	X	X

Roles (i.e. . Analyst, Developer, Tester)	End-user, IT team
Related Methodologies, Best Practices and Tools	Authorisation, authentication
Related Technologies (i.e. Java)	JAVA/JAVA EE

FOSS Communities Using This Library	OWASP	N/A	OpenSSL	N/A	Jenkins	N/A
	OpenStack	N/A	PIWIK	N/A	Eclipse	N/A
	Spring FW	N/A	Red Hat	X	Bitergia	N/A
	LibreOffice	N/A	Drupal	N/A	OwnCloud	N/A
	Debian	X	Apache Tomcat	N/A		

LB&B5	**Tool Name:**	**Spring Security**

Use	**Objectives**	**Benefits**
Authentication and access control framework	It is a framework that focuses on providing both authentication and authorisation to Java applications.	Provides authentication, authorisation and other security features for enterprise applications

SDLC Phase Where It Is Used					
Analysis	Design	Development	Testing	Deployment	Maintenance
N/A	X	X	X	X	X

Roles (i.e. . Analyst, Developer, Tester)	Analyst, Architect, Developer, Tester.
Related Methodologies, Best Practices and Tools	
Related Technologies (i.e. Java)	Java

FOSS Communities Using This Library	OWASP	N/A	OpenSSL	N/A	Jenkins	N/A
	OpenStack	N/A	PIWIK	N/A	Eclipse	N/A
	Spring FW	X	Red Hat	N/A	Bitergia	N/A
	LibreOffice	N/A	Drupal	N/A	OwnCloud	N/A
	Debian	N/A	Apache Tomcat	N/A		

LB&B6	Tool Name:	OpenStack Identity

Use	Objectives	Benefits
Security module.	Increase the security with the provision of a security module.	Ir provides a authentication and authorization functions from front and back ends servers.

SDLC Phase Where It Is Used					
Analysis	Design	Development	Testing	Deployment	Maintenance
X	X	N/A	N/A	X	N/A

Roles (i.e. . Analyst, Developer, Tester)	Analyst
Related Methodologies, Best Practices and Tools	
Related Technologies (i.e. Java)	

FOSS Communities Using This Tool	OWASP	N/A	OpenSSL	N/A	Jenkins	N/A
	OpenStack	X	PIWIK	N/A	Eclipse	N/A
	Spring FW	N/A	Red Hat	N/A	Bitergia	N/A
	LibreOffice	N/A	Drupal	N/A	OwnCloud	N/A
	Debian	N/A	Apache Tomcat	N/A		

3.5. Programming Languages Used by the Analysed FOSS Communities During the Software Development Lifecycle

LG1 Language Name:	**Java**

Use	**Objectives**	**Benefits**
A general-purpose software programming language	Provides concurrent, class-based, object oriented programming, and is specifically designed to have a few implementation dependencies as possible	Java code can run on all platforms that support Java without the need for recompilation

SDLC Phase Where It Is Used				
Analysis	Design	Development	Testing	Deployment
N/A	X	X	X	X

Roles (i.e. Analyst, Developer, Tester)	Developer, Tester, Analyst, Architect
Related Methodologies, Best Practices and Tools	Object Oriented
Related Technologies (i.e. Java)	

FOSS Communities Using This Language	OWASP	N/A	OpenSSL	N/A	Jenkins	X
	OpenStack	N/A	PIWIK	N/A	Eclipse	X
	Spring FW	X	Red Hat	N/A	Bitergia	N/A
	LibreOffice	X	Drupal	N/A	OwnCloud	N/A
	Debian	N/A	Apache Tomcat	X		

LG2	**Language Name:**	**PHP**

Use	**Objectives**	**Benefits**
General-purpose software programming language	Programming language originally designed to create web sites	Flexible and powerful, especially when creating web sites

SDLC Phase Where It Is Used				
Analysis	Design	Development	Testing	Deployment
N/A	X	X	X	X

Roles (i.e. Analyst, Developer, Tester)	Developer, Tester, Analyst, Architect
Related Methodologies, Best Practices and Tools	Object Oriented
Related Technologies (i.e. Java)	

FOSS Communities Using This Language	OWASP	N/A	OpenSSL	N/A	Jenkins	N/A
	OpenStack	X	PIWIK	X	Eclipse	N/A
	Spring FW	N/A	Red Hat	N/A	Bitergia	N/A
	LibreOffice	N/A	Drupal	X	OwnCloud	X
	Debian	N/A	Apache Tomcat	N/A		

LG3 Language Name:			**Python**			
Use		**Objectives**		**Benefits**		
Multi-paradigm programming language		Serves as a scripting language for web applications		It is a structured object-oriented programming language that is fully supported, with a number of features for functional and aspect-oriented programming		
SDLC Phase Where It Is Used						
Analysis	Design	Development	Testing	Deployment		
N/A	X	X	X	X		
Roles (i.e. Analyst, Developer, Tester)		Developer, Tester, Analyst, Architect				
Related Methodologies, Best Practices and Tools		Object-Oriented, Structured Programming				
Related Technologies (i.e. Java)						
FOSS Communities Using This Language	OWASP	N/A	OpenSSL	N/A	Jenkins	N/A
	OpenStack	X	PIWIK	N/A	Eclipse	N/A
	Spring FW	N/A	Red Hat	N/A	Bitergia	N/A
	LibreOffice	N/A	Drupal	N/A	OwnCloud	N/A
	Debian	X	Apache Tomcat	N/A		

LG4 Language Name:						
C						
Use		**Objectives**		**Benefits**		
General-purpose and imperative computer programming language.		Provides constructs that map efficiency to typical machine instructions		It is a flexible language that provides a multi-programming styles. It is a highly transportable language.		
SDLC Phase Where It Is Used						
Analysis	Design	Development		Analysis		Design
N/A	X	X		N/A		X
Roles (i.e. Analyst, Developer, Tester)		Developer, Tester, Analyst, Architect				
Related Methodologies, Best Practices and Tools						
Related Technologies (i.e. Java)						
FOSS Communities Using This Language	OWASP	N/A	OpenSSL	X	Jenkins	N/A
	OpenStack	N/A	PIWIK	N/A	Eclipse	N/A
	Spring FW	N/A	Red Hat	X	Bitergia	N/A
	LibreOffice	N/A	Drupal	N/A	OwnCloud	N/A
	Debian	X	Apache Tomcat	N/A		

LG5 Language Name:						
C++						
Use		**Objectives**		**Benefits**		
General-purpose programming language		It is a light-weight abstraction programming language for building and using efficient and elegant abstractions.		It provides a very high control for the programmer. It provides a performance efficiency and flexibility of use as its design highlights		
SDLC Phase Where It Is Used						
Analysis	Design	Development		Analysis	Design	
N/A	X	X		N/A	X	
Roles (i.e. Analyst, Developer, Tester)		Developer, Tester, Analyst, Architect				
Related Methodologies, Best Practices and Tools						
Related Technologies (i.e. Java)						
FOSS Communities Using This Language	OWASP	N/A	OpenSSL	N/A	Jenkins	N/A
	OpenStack	N/A	PIWIK	N/A	Eclipse	N/A
	Spring FW	N/A	Red Hat	X	Bitergia	N/A
	LibreOffice	X	Drupal	N/A	OwnCloud	N/A
	Debian	X	Apache Tomcat	N/A		

LG6 Language Name:						
Ruby						
Use		**Objectives**		**Benefits**		
General purpose programming language		Generation of a flexible language for the programmers		It provides a dynamic type system automatic memory management.		
SDLC Phase Where It Is Used						
Analysis	Design	Development		Analysis	Design	
N/A	X	X		N/A	X	
Roles (i.e. Analyst, Developer, Tester)		Developer, Tester, Analyst, Architect				
Related Methodologies, Best Practices and Tools						
Related Technologies (i.e. Java)						
FOSS Communities Using This Language	OWASP	N/A	OpenSSL	N/A	Jenkins	X
	OpenStack	N/A	PIWIK	N/A	Eclipse	N/A
	Spring FW	N/A	Red Hat	N/A	Bitergia	N/A
	LibreOffice	N/A	Drupal	N/A	OwnCloud	N/A
	Debian	N/A	Apache Tomcat	N/A		

4 Analysis of Identified Software Development Methodologies Used in FOSS Communities

In order to conduct this analysis, the following FOSS communities were identified:

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.

Additionally, the following communities were interviewed but not analysed:

1. Free Software Foundation Europe (FSFE), one of the main open source hubs in Europe. This organisation does not have any software development projects; however, they are willing to provide support in anything related to the FOSS communities, i.e. new contacts, general documentation, etc.
2. Mozilla, one of the most popular FOSS communities that develops open source software such as “Firefox”, and “Thunderbird”. We interviewed security experts who are involved in code review for

Mozilla projects, they have delegated this task to an external private company and it was not possible to get recommendations and best practices about this field from their side.

Out of the 14 + 2 FOSS communities selected, we interviewed:

1. Apache – tomcat
2. Drupal
3. Free Software Foundation
4. Libre Office
5. OWASP Community

for the remaining 9 communities, everis´ teams conducted the information gathering process by researching the communities documentation found on their websites, and also wikis, code repositories, and forums.

4.1.Project Management

4.1.1.Methodologies

This section analyses the different project management methodologies used in FOSS communities, divided according to the formality of the methodology used. Under this division criterion, a methodology is considered 'formal' when it is the implementation of a standard one, and 'informal' when it is a custom made methodology, or just a set of processes, guides, teams and functionalities that are defined and well documented.

Figure 2: Project Management Approach

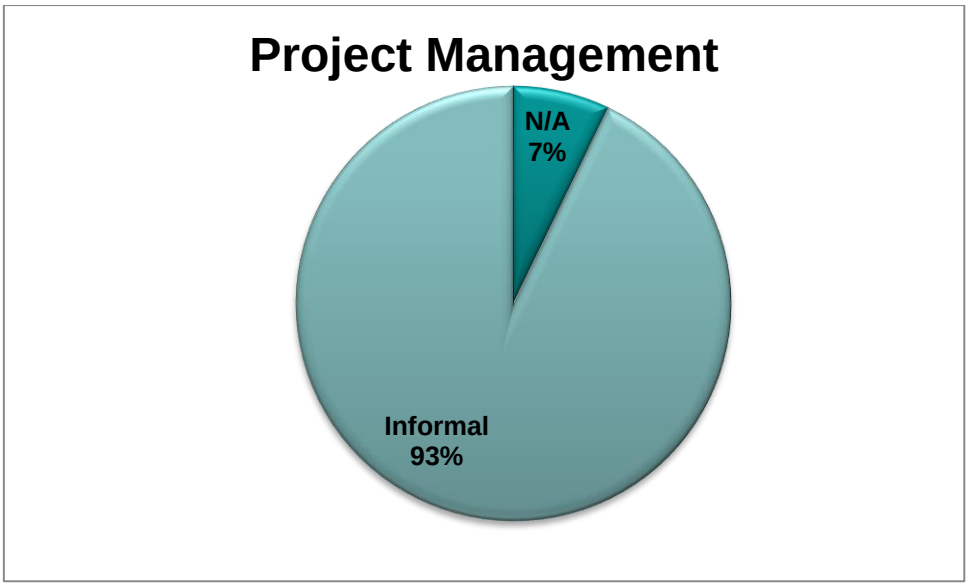

Table 1: Project Management Approach

Project Management Methodology Approach	Out of the 14 Analysed Projects	
	Number of Communities	Percentage of the Communities
Informal	12	93%
N/A	2	7%

Most FOSS communities are based on volunteers, according to their interpretation of 'open', so anyone can collaborate with the community. Nevertheless, they are aware of bad quality or malicious contributions so in order to prevent these, they have defined mitigation actions such as the following:

1. The first rule that communities apply is to review any contribution made, regardless of the originating source. To ease the revision of contributions, they produce guides on how the work should be done. Most of these guides address different types of contributions, such as: software development, documentation writing, user support, legal assistance etc. Most of FOSS communities have special security groups that manage and organise projects related to software security, and some of them also have a legal group that manages license issues.
2. Some of them use a more formal methodology in which documentation regarding processes and community management is provided. They create several work groups and communication channels in order to be more effective. Some of them assign 'mentors' to new contributors, who are experienced collaborators that guide newcomers.
3. The majority of FOSS communities have different member levels, where critical actions such as the inclusion of contributions, are performed by trustworthy people. Most have a board of directors that decide about the strategy of the FOSS community, and project management committees/groups that manage the community projects. These management groups are comprised by members who are most involved in the project.
4. Some of these communities have additional profiles for contributors, based on the required level or responsibilities that need to be assigned to them. Such profiles are assigned to those who have been promoted due to their merits. This organisational method follows a meritocratic approach where merits are considered the main promotion mechanism.
5. In the case of FOSS communities that have been created by a private organisation, the organisation occasionally decides on issues affecting the community and/or its projects. As a result, private sponsors are in the board of directors, and participate in the definition and direction of the community's strategy.
6. Some FOSS communities have foundations that represent them as a legal entity. This provides several benefits for such communities and their sponsors. Sponsors can receive tax benefits for supporting non-profit organisations, and communities can employ full time workers.

4.1.2.Conclusion

Most of the FOSS communities' project management methodologies in use are informal, due to the fact that FOSS communities are mainly comprised by volunteers. This means that critical resources such as time, budget and workforce are limited. Furthermore, the majority of volunteers seem to be attracted mainly to software development with only a few of them getting involved in project management or documentation writing. Nevertheless, large communities, and those that have strong relations with private organisations, present a higher maturity level in project management as a result of having increased resources (i.e. budget, time and/or workforce).

4.2.Software Development Lifecycle

4.2.1.Software Development Lifecycle Methodologies

4.2.1.1.Methodologies

The existing software development methodologies in FOSS communities will be analysed in this part of the document. A methodology is considered 'formal' when it implements a standard (e.g. Scrum, Agile, etc.), and 'informal' when it uses its own methodology.

Figure 3. Software Development Management Approach

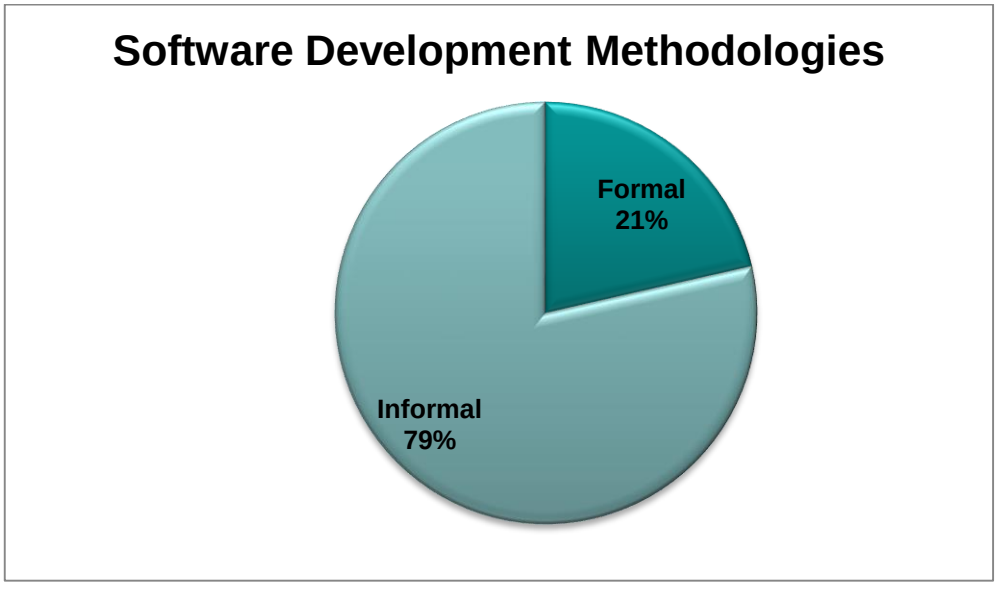

Table 2: Software Development Management Approach

Development Methodology	Out of the 14 Analysed Projects	
	Number of Communities	Percentage of the Communities
Informal	11	79%
Formal	3	21%

The methodology of software development is influenced by the same factors we mentioned further above for project management. FOSS communities are comprised mainly by volunteers, which makes it hard to apply standard software development methodologies. However, some special cases enable the use of standard software methodologies.

The different approaches in dealing with software development methodologies are listed below:

1. FOSS communities that have human resources assigned from their sponsors can use standard methodologies. These contributors are professional software developers that have the time to follow software development methodologies.
2. Some FOSS communities have non-profit foundations with full-time personnel, and can dedicate resources to coordinate software development that follows standard methodologies.
3. If the FOSS community has been created from a private organisation, this private organisation can dedicate resources to software development and easily follow standard methodologies.
4. Apart from the above cases, the majority of FOSS communities do not follow any standard software development methodologies; their approach is more informal and includes:
    - Leaders that coordinate the objectives. These objectives or needs are publicised among the community and contributors start to work on them.
    - In order to manage these objectives or needs, some FOSS communities use a bug-tracking platform or code repository. On these platforms, the list of needs is available for browsing and lookup, and community members can submit their work to satisfy them.
    - Any contributor can select any work task and work on it; when the work is finalised, the contribution is reviewed according to the community review procedure.

One FOSS community has recommended the concept called ‘SecDevOps’, which is the combination of the terms ‘Security’ and ‘DevOps’. This practice ensures that security is carefully implemented from the beginning, implementing security measures at every stage of software development. These actions ensure software dependability, trustworthiness and resilience.

4.2.1.2. Tools

- GitHub
- Launchpad
- Bugzilla

- OpenHUB
- Jira
- FusionForge (Alioth)

4.2.1.3. Conclusion

Most of the FOSS communities are volunteer-based. This results in a heterogeneous software development process thanks to the diversity of volunteers, where most of the contributions come from individual persons. Consequently, software development methodologies are quite varied and, in some cases, are based on team leaders' opinions. The role of private organisations and foundations in software development is very important, since they can change how software development is being conducted, by using standard methodologies.

Additionally, most of the FOSS communities use tools to manage software development. They usually are integrated with the project code repository, such as GitHub, Launchpad, OpenHub. These tools provide an efficient manner to deal with tracking and coding management tasks on the same platform.

4.2.2. Security Definition

4.2.2.1. Security Requirements

Security requirements are a relevant concept to study in software development because of their impact on software security. The security level of software depends on how and where security has been considered during the software development process, and it also impacts the resource allocation in order to secure the software. Explicit security requirements are the most efficient ways to incorporate security at the beginning of the Software Development Lifecycle.

Figure 3: Security Requirements in FOSS Communities

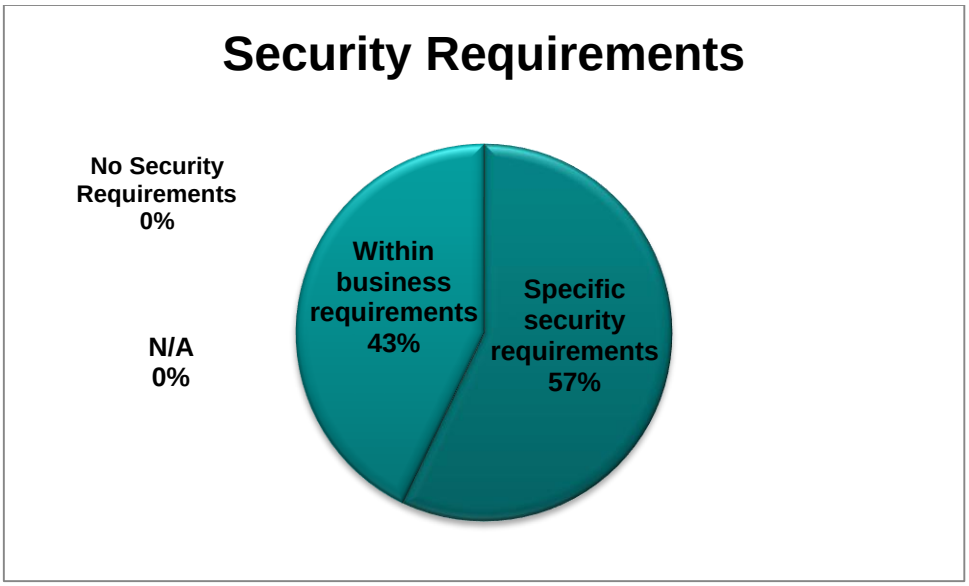

Table 3: Security Requirements in FOSS Communities

Security Requirements	Out of the 14 Analysed Projects	
	Number of Communities	Percentage of the Communities
Specific Security Requirements	8	57%
Within Business Requirements	6	43%
No Security Requirements	0	0%
N/A	0	0%

As far as security requirements are concerned, they are analysed in different ways, depending on their awareness of security, which in turn is related to the community's maturity level.

When a FOSS community is aware of security and they have specific security requirements, it usually is a mature community.

Some factors seem to be directly related to whether specific security requirements are defined or not. Some relevant factors are the size of the community, the usage or not of FOSS and how critical the software is. However, the main factor that seems to trigger the definition of security requirements is whether there have been previous security incidents that led to an increase in security awareness in the FOSS communities.

FOSS communities take into account security requirements in different ways:

1. Security requirements are analysed specifically, describing the security needs of the FOSS.
2. Security requirements are considered as business requirements, as a consequence of two different points of view:
    - Security is an inherent part of the software's functionality, so it is not necessary to add specific requirements.
    - Security is not considered in software design, so there are no specific security requirements, although these can be added in subsequent phases.

4.2.2.2. Security Awareness

Another security concept that shows the maturity level of a software project's security level is the execution of a threat modelling. This security exercise provides information and awareness of security risks that the software poses or might be exposed to, and it also provides controls to mitigate them. When a software development project incorporates threat modelling, the software is ready to deal with common security risks, minimising the attack surface.

Figure 4: Security Definition Phase in FOSS Communities

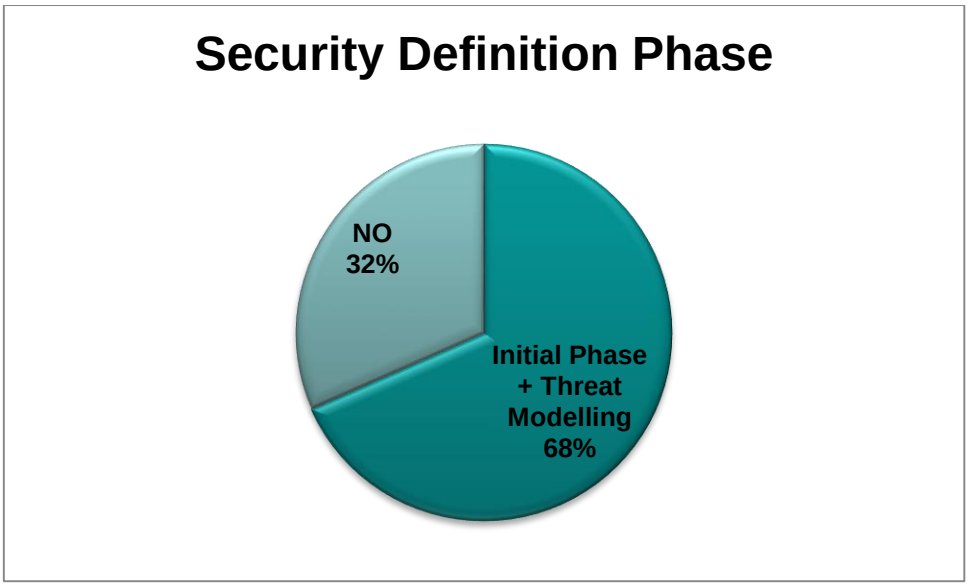

Table 4: Security Definition Phase in FOSS Communities

Security Definition	Out of the 14 Analysed Projects	
	Number of Communities	Percentage of the Communities
Initial Phase + Threat Modelling	8	68%
No	6	32%

This security analysis is strongly related to the assessment of security risks, which is the first step to awareness of potential security issues that the FOSS communities must overcome.

Table 5: Execution of Risk Assessment in FOSS Communities

Risk Assessment	Out of the 14 Analysed Projects	
	Number of Communities	Percentage of the Communities
Yes	12	86%
No	2	14%

The execution of risk assessment and threat modelling are the main FOSS community tools to evaluate the risk level of an application; this process should be strongly related to the set of security requirements that need to be satisfied. As we previously explained, this set of security requirements can be specific, or it may be included within the business requirements.

The way that FOSS communities carry out this security exercise can vary depending on their specific context.

- This exercise is conducted by a specific security team, and it is disseminated in the community in order to promote security awareness. This usually happens in communities with a high level of security awareness.
- Some of them create guidelines for FOSS developers, satisfying the security needs (requirements) of the community. This is also used during contribution reviews, to accept software contributions.
- In some communities, security teams develop tools to assess the security of contributions.

Moreover, most of the FOSS communities provide security information to developers and FOSS users by means of guidelines or wiki pages for awareness and to deploy countermeasures against security risks.

4.2.2.3. Conclusion

In order to measure the security level of a software development project, security requirements are a key factor to take into account. Among the FOSS communities, different maturity levels can be found, given their diversity. However, security is added in FOSS software depending on several aspects such as the criticality of the software, the FOSS community size and longevity, and FOSS usage.

The size of the FOSS community also affects the number of security volunteers that work on the security aspects of the software under development.

The trend obtained after the analysis indicates that FOSS communities provide developers and users with information on how to respond to security issues, as well as how to mitigate them during the development or configuration.

Some of the FOSS communities have dedicated security teams that efficiently conduct security risk assessments.

Another aspect that improves the security in FOSS is having strong relationship with private organisations, since they could have concerns regarding the security of the FOSS. These concerns could be transformed into efforts to improve security, especially if the FOSS software is an important element of their business strategy.

4.2.3. Testing and Validation

Testing is performed in order to find possible bugs or vulnerabilities before the application release; it also helps to ensure software quality and that the expected requirements have been implemented. In FOSS communities this phase has special importance due to their nature, in which code review is performed systematically.

4.2.3.1. Automatic Testing

Automatic testing allows the execution of written tests, mostly in silent (unattended) mode, without the manual intervention of the Development or QA teams. This approach is conducted by software testing suites that are comprised by groups of individual tests which are logically-related, incremental, and repeatable. This analysis encompasses functional, non-functional, unit and regression tests. In FOSS communities this method of testing allows for an efficient way of using the limited community resources.

Table 6: Automatic Testing

Automatic Testing	Out of the 14 Analysed Projects	
	Number of Communities	Percentage of the Communities
Yes	12	86%
No	2	14%

Automatic testing is an efficient way to save community resources and check software. This is carried out differently among FOSS communities:

1. Some of them use automatic integration, which performs some tests automatically.
2. Some of them use automatic static code analysis to find possible issues.
3. Some of them use unit testing tools to launch sets of tests.

4.2.3.2. Security Testing

This section analyses the execution of security tests to identify security bugs or vulnerabilities. These tests use techniques like penetration tests, vulnerability scans and black and/or white box testing. In FOSS communities, secure testing is conducted by multiple reviewers working on code review before the final acceptance; as a result, some code errors and Easter eggs can be resolved at this stage. This is one of the strongest points of FOSS, the participation of many reviewers during code checking.

Table 7: Security Testing

Security Requirements	Out of the 14 Analysed Projects	
	Number of Communities	Percentage of the Communities
Software Review	13	93%
Vulnerability Assessment	7	50%
Penetration Testing	1	7%
N/A	1	7%

According to the results, most FOSS communities perform some security tests and, in general, are efficient at conducting software reviews.

Some security tests are not feasible, because they require a security specialist. In order to overcome this issue, communities follow different approaches such as:

1. Having a security specialist among their volunteers.
2. Using the available community resources and contacting a security service to conduct the testing.
3. Testing is carried out by some public administration or private organisation, and the results are shared in public forums.

Also, some tests require having the software up and running, and for some FOSS communities this is not possible due to the lack of resources (people and infrastructure), so the running software has to be tested in other environments. Communities with larger resources can do this, especially if they have strong relations with private organisations.

4.2.3.3. Validation Testing

In FOSS communities, users can validate the code while testing the application's functionality. In most communities, FOSS is not running on the community servers, so users have to deploy it locally by themselves in order to perform validation tests. Their feedback can be easily communicated to FOSS developers thanks to the existing communication channels of FOSS communities.

4.2.3.4. Tools and Methods

- OWASP ESAPI
- OWASP Zed Attack Proxy
- OWASP AppSensor
- OpenStack Bandit
- OpenStack Anchor
- OpenStack Zuul
- Gerrit
- Coverity
- OpenVas
- Rats
- Flawfinder
- SonarQube

4.2.4. Release Management

In FOSS communities the release management process is different due to the fact that there is no actual deployment. They deliver software releases and FOSS users deploy them.

4.2.4.1. Conclusion

Most FOSS communities are mature in terms of testing and validation as a result of several factors:

1. Sustainability of the FOSS depends on the quality and security of the software.
2. Wide usage of FOSS relies on software trust. This implies that software testing is critical.
3. Private organisations that do business with FOSS, need to ensure its trustworthiness to be able to offer their services. This factor drives the testing process in the FOSS communities.

4.2.4.2. Release Planning

Most FOSS communities have a release plan where new functionalities, resolved bugs, and other improvements are included. Most communities include security in their roadmap, giving the opportunity to add security improvements in future releases.

Table 8: Roadmap in FOSS Communities

Roadmap in the FOSS Communities	Out of the 14 Analysed Projects	
	Number of Communities	Percentage of the Communities
Yes, including security	8	57%
Yes, without security	4	29%
No	2	14%

Most FOSS communities have a release plan to deliver new releases. The frequency differs among the communities, but the trend indicates that there is a plan for new releases that can be executed in several ways:

1. A planned way, where the release is delivered with all implemented modifications being up-to-date.
2. A planned way, when a new version of the underlying technology of the FOSS is delivered.
3. An informal way, when the community considers that they have made enough modifications and a new release is needed.
4. After a critical bug, where the new release contains the fix for the critical bug or vulnerability.

4.2.4.3. Continuous Testing and Validation

Continuous testing is the process of executing automated tests as part of the delivery pipeline. The focus is on receiving continuous feedback regarding the business risks related to a software release candidate and determining if the software is ready to be promoted through the delivery pipeline. During this process, functional and non-functional tests, static code analysis and security testing may be involved.

Continuous integration and testing is used in most FOSS communities to verify the stability of the software, with the additional benefit of saving community resources. In most cases, they use the software tool called 'Jenkins' to fulfil this task.

Table 9: Continuous Integration in FOSS Communities

Continuous Integration	Out of the 14 Analysed Projects	
	Number of Communities	Percentage of the Communities
Yes	12	86%
No	0	0%
N/A	2	14%

As previously explained, continuous testing is used as another way to do automatic testing. This process automatically integrates new code, does validation tests, and prepares the software to be released, thus saving community resources and improving the quality of the software.

Some communities integrate unit testing in the continuous integration environment as a way of conducting additional tests.

4.2.4.4. Channels and Tools Used

- JUnit
- Apache Gump
- Jenkins
- OpenStack Zuul
- Gerrit

4.2.4.5. Conclusion

Most FOSS communities follow a release process, containing a release planning that includes security and continuous integration. Release management in FOSS communities can be considered a mature aspect that enables the delivery of quality software.

Additionally, some FOSS communities provide different types of releases, depending on the software's stability. This in turn provides flexibility to the FOSS users who can decide between stability or cutting-edge functionality.

4.2.5. Inspection and Code Review

As indicated in previous sections, this aspect of the FOSS communities is critical. Inspection and code review are mature processes that aim to provide quality and security to FOSS. These features are essential for FOSS usage, and consequently for the sustainability of FOSS communities.

4.2.5.1. Code Review

In this section we will analyse the code review process of FOSS communities, considering who reviews the code, and where in the SDLC it is done.

The code is reviewed before receiving the acceptance from different reviewers. Moreover, it can be reviewed at any time, thanks to the open nature of FOSS. It is important to highlight the fact that in most of the FOSS communities, the results of code reviews are public.

Table 10: Code Review in FOSS Communities

Who Reviews the Code	Out of the 14 Analysed Projects	
	Number of Communities	Percentage of the Communities
Community Members	11	79%
Specific Team	5	36%
N/A	2	15%

Code review is a necessary process to ensure software quality, and it is done in all communities. Nevertheless, this differs among communities:

- In most FOSS communities, the code is reviewed by other community members.
- In some communities, the code is analysed by a specific team before being integrated. This team is formed by trustworthy community members.

4.2.5.2. Tools

- GitHub
- Launchpad
- OpenHUB

4.2.5.3. Projects Reviewed by Security Experts

The secure code implementation can be reviewed by security experts to try to find possible security issues. The following table presents the data obtained from the analysis, showing that all FOSS communities have their code reviewed by security experts in one way or another.

Table 11: Table Regarding Security Experts Review in FOSS Communities

Security Experts Review in FOSS Communities	Out of the 14 Analysed Projects	
	Number of Communities	Percentage of the Communities
Yes	8	75%
No	6	25%

4.2.5.4. Phase Where the Project is Reviewed by Security Experts

According to the SDLC, the code can be reviewed in different phases as shown in Table 12.

Table 12: Phase Where Security Experts Review the Code in FOSS Communities

Phase Where Security Experts Conduct Code Review in FOSS Communities	Out of the 14 Analysed Projects	
	Number of Communities	Percentage of the Communities
All Phases	2	15%
Initial Phases	0	0%
During Development	5	35%
At the End	4	37%
N/A	5	35%

In most FOSS communities, security experts review the code to guarantee a certain level of security in the software; however, this can be conducted in different phases:

1. Security review is rarely implemented in all software phases.
2. In some communities, security review is conducted during the development of each contribution.
3. In some communities, the security is reviewed at the end of the development but before delivering the new release.

4.2.5.5. Conclusion

Code review is a mature process in any FOSS community, mainly because all contributions have to be reviewed before being accepted. Most FOSS communities follow the rule that code has to be reviewed by different developers/contributors. Moreover, they have a specific team of security experts to review the code, and conduct automatic security tests. These reviews are conducted mainly during the development phase of the SDLC and at the end of the development, before it becomes a candidate release.

The results of these reviews are public, so any FOSS user can check the software development and validation processes, and also participate in the review to find possible security issues before using the software.

Some FOSS communities use security bug-hunter portals (e.g. HackerOne) to improve their security. This practice requires financial resources that must be provided by the project itself. Therefore, those FOSS communities that have strong relations or support from private organisations have a better chance of obtaining such resources and taking advantage of these portals to improve their security.

Last, there is the human factor to take into account when dealing with code review: the reputation of the developer or contributor. Because the contributor's work is tested, a bad quality code can largely impact his position in the community.

4.2.6. Application Authentication and Authorisation

As far as security is concerned, authentication and authorisation are critical aspects to avoid unauthorised information disclosure and privilege escalation. In this section the software modules that provide functionality for authentication and authorisation are analysed, focusing on where these modules are developed. Furthermore, the authorisation model is examined to determine which one is preferred by FOSS communities.

4.2.6.1. Authentication

This section analyses the authentication modules used by FOSS communities. We focused on the usage of the authentication module, highlighting where the module has been developed. The different options for the development of the authentication module are as follows:

1. Developed internally (within the project).
2. Developed outside the project, but within the FOSS community.
3. Developed outside the FOSS community.

The different options to develop the authentication module are shown in Table 13.

Table 13: Authentication Modules in FOSS Communities

Authentication Modules in FOSS Communities	Out of the 14 Analysed Projects	
	Number of Communities	Percentage of the Communities
Developed internally (within the project)	6	43%
Developed outside the project, but within the FOSS community	3	22%
Developed outside the FOSS community	6	43%
N/A	4	29%

4.2.6.2. Authorisation

For authorisation we considered these models: 'Role-based Access Control' (RBAC), 'Mandatory access control' (MAC), and 'Discretionary Access Control' (DAC). Furthermore, user groups are also considered in this study.

The use of the different authorisation models is shown in Table 14.

Table 14: Authorisation Model in FOSS Communities

Authorisation Model in FOSS Communities	Out of the 14 Analysed Projects	
	Number of Communities	Percentage of the Communities
RBAC	8	57%
MAC	3	21%
DAC	3	21%
User Groups	4	29%
N/A	5	36%

4.2.6.3. Conclusion

Some FOSS communities develop their own authentication and authorisation modules, instead of relying on third-party components. In contrast, some of them rely on external FOSS components.

This second approach aims to use a common well tested module for authentication, and to provide information about cooperation among FOSS communities.

As for authorisations, Role-based Access Control is the most used model among the analysed FOSS communities, where privileges are assigned according to the predefined roles and privileges are easily managed, avoiding customised cases for users. One third of FOSS communities complement their access control models by user groups.

It is worth mentioning that FOSS communities use known access control models, avoiding the use of their own access control models.

4.3.Project Maintenance

The importance of quality and security in FOSS has been discussed in previous sections; however some additional processes are required to attain it. These processes are divided in two categories: Incident Management and Problem Management.

The first one addresses how to react when a security issue arises, while the second one aims to study how software bugs are identified and solved.

4.3.1.Incident Management

4.3.1.1.Incident Resolution

Once a security issues arises, the FOSS communities have a predefined process to deal with it. In FOSS communities, the discovery of a vulnerability is considered and incident, and it is managed using one of the platforms shown in Table 15 below.

Table 15: Incident Resolution in FOSS Communities

Incident Resolution in FOSS Communities	Out of the 14 Analysed Projects	
	Number of Communities	Percentage of the Communities
Email box	9	64%
Bug management platform (e.g. Bugzilla)	11	78%
N/A	1	7%

Most FOSS communities deal with incidents on a daily basis, and they have different ways to coordinate the incident resolution:

- Using a bug platform to manage the different bugs or vulnerabilities, and to coordinate the actions to solve it.

- For critical vulnerabilities, they usually have a set of email contacts to report the issue to. They follow this process in order to limit the number of people that know about the vulnerability. This information is managed by a special security team that solves the issue.

4.3.1.2.Handling of Major Incidents

According to the definition of an incident used in the previous section, a major incident is a critical vulnerability that can compromise the FOSS. The results in this analysis indicate that all communities raise the priority to solve major issues.

4.3.1.3.User Notification

This section addresses how these incidents are notified to FOSS users. The results present the different communication channels used by FOSS communities.

Table 16: User Notification Channels Used in FOSS Communities

User Notification in FOSS Communities	Out of the 14 Analysed Projects	
	Number of Communities	Percentage of the Communities
FOSS Main Website	11	79%
Email List	4	29%
Release Notes	6	43%
JIRA	2	15%
N/A	1	7%
Silent Mode	0	0%

FOSS users are always notified about incidents or vulnerabilities by the FOSS communities, once the remediation is available, using one of 4 channels:

1. Most FOSS communities use their own webpage to publish this information, and it is the main channel to exchange information with the users.
2. Some of them also use the wiki to explain the issue and the remediation in depth.
3. Some of them also utilise an email list to inform FOSS users immediately.
4. In some FOSS communities, they provide a release note with the same goals as the wiki page.

4.3.1.4.Conclusion

As far as incident response is concerned, most FOSS communities have a standard process to deal with security incidents. This process usually uses a platform to manage vulnerabilities and bugs (e.g. Bugzilla), whereas a different email address list is used for critical ones.

Different procedures are conducted depending on the severity of the vulnerability and/or bug, as well as what resources to utilise to solve it. In most cases, the task priority is raised and the resolution is delivered in the form of a release.

For user notification, most FOSS communities use their webpages as the main channel to communicate bugs, whereas some of the communities also utilise email lists, wikis or release notes to inform the FOSS users.

4.3.2. Problem Management

4.3.2.1. Identification of Security Updates or Bugs

This section analyses how FOSS communities identify potential security updates or bugs in advance, as a preventive measure. This is important since newly discovered vulnerabilities or bugs handled on time will reduce the likelihood of service disruptions, and thus will ensure the trustworthiness of the software.

Table 17: Methods for Identifying Bugs in FOSS Communities

Method for Bug Identification in FOSS Communities	Out of the 14 Analysed Projects	
	Number of Communities	Percentage of the Communities
Regression Test	12	86%
Code Review	11	79%
FOSS Users	13	94%
External Security Experts	3	22%
N/A	1	7%

FOSS communities have different methods for detecting potential bugs or vulnerabilities, which also depends on the development phase:

1. Using regression tests to identify if the software has a bug. It is usually performed by means of automatic testing and continuous integration testing.
2. Many bugs are detected and corrected during the review of the contributions provided by community members.
3. FOSS users detect many issues while using the software, and some of them may be security related. If they do special security testing, they can provide the results to the community and help them improve the FOSS.
4. When security experts review the project, they may detect risks that could potentially compromise its security.

4.3.2.2. Problem Resolution Plan

This section focuses on how FOSS communities solve bugs or vulnerabilities. Once a FOSS community detects a bug or vulnerability, it tries to solve it according to the criticality. If the error is detected in the candidate release, it is corrected before the delivery process starts. If it is detected in the current release, the process varies: if the bug is a minor issue, it will be corrected in the following release; if the bug or vulnerability is a major incident, then a new release or patch will be developed and deployed as soon as possible.

The analysis shows that all communities have a special process to fix bugs, and the most common way to deliver the remediation is by deploying a new release.

4.3.2.3. Tools and Resources Used

- Bugzilla
- Launchpad
- GitHub
- OpenHUB

4.3.2.4. Conclusion

FOSS communities are mature in terms of dealing with bugs and vulnerabilities. This can be noticed in the process that they follow to identify bugs or vulnerabilities, the resolution process, and the notification process to the FOSS users.

Most FOSS communities have special processes to solve critical issues, and they deliver the solution by means of new releases.

It is important to highlight that communities with strong relations to private organisations are more efficient when it comes to solving bugs or vulnerabilities. Due to the fact that these organisations use FOSS at the core of their business, they use their own resources to solve security issues. In some communities, their contacts for security issues are private organisation employees.

4.4. FOSS Communities, Private Organisations and European Institutions

This section of the document shows the different types of collaboration between FOSS communities and external organisations. As explained in previous sections, FOSS communities are affected by these relations. During this study, said interactions have been analysed, and the results can be observed in the following table.

Table 18: Enterprise Collaboration in FOSS Communities

Enterprise Collaboration Activities in FOSS Communities	Out of the 14 Analysed Projects	
	Number of Communities	Percentage of the Communities
Software Development Contribution	10	72%
Software Review Contribution	5	36%
Security Review Contribution	5	36%
Sponsors	11	79%
Donations	9	65%

Most FOSS communities are involved in one or more of the collaborations shown in Table 18. These collaborations are achieved in the form of software contribution, software review, financial sponsorship, IT infrastructure and other kind of donations. The extent of the collaboration has a direct impact on software quality, software security and community sustainability and, thus, FOSS-reliance is significantly increased.

There are different reasons that explain why these collaborations take place:

1. The organisation does business using FOSS (consulting, commercial support, training, etc.)
2. The organisation uses FOSS for critical business processes, or for security issues.
3. The organisation receives new software functionality or components from the FOSS community, that can be incorporated in its own software.

Regarding the collaboration with European Institutions, most FOSS communities see European Institutions (EUI) as a main driver for FOSS usage. They think that EUI should support FOSS communities, especially when EUI use a FOSS component. This is also beneficial for EUI, since FOSS is ‘supported’ by a live FOSS community.

4.5.Relevant Opinions and Advice from Interviewees

This final section gathers some additional contributions provided during the interviews:

1. Every FOSS community works following its own software development methodology. Information exchange should be promoted for the benefit of sharing knowledge, experience and best practises.
2. FOSS communities have developed reactive and mature processes to respond to known vulnerabilities, which should be enhanced by a preventive approach.
3. Threat Modelling is highly recommended.
4. Security should be considered from the beginning, instead of applying it after or during development.
5. European Institutions should support FOSS communities and, more importantly, software security communities, with a focus on the ones which develop the software that European Institutions use. One possibility is for the European Institutions to do code reviews and share the results with the affected communities.
6. European Institutions should share the benefits of the FOSS they use with other member states, as well as the list of FOSS they use.

5 References

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

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

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

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

6 Annexes

6.1.Questionnaires for the Interview

Questionnaire FOSS communities

Questionnaire experts

6.2.Executive Summary

Executive summary

