WP2

DIGIT B1 - EP Pilot Project 645

Deliverable 11: **Design of the Method for Performing the Code Reviews for the European Institutions**

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

May 2016

DIGIT Fossa WP2 – Governance and Quality of Software Code – Auditing of Free and Open Source Software.

Deliverable 11: Design of the method for performing the code reviews for the European institutions

Author:

Disclaimer

Contents

List of Tables

List of Figures

ACRONYMS AND ABBREVIATIONS

CVSS	Common Vulnerability Scoring System
CWSS	Common Weakness Scoring System
DDoS	Distributed Denial of Service
DG	Directorate General
DoS	Denial of Service
EI	European Institutions
EP	European Parliament
FOSS	Free and Open Source Software
FOSSA	Free and Open Source Software Auditing
OS	Operating System
SDLC	System Development Life Cycle
SEO	Search Engine Optimization
WP	Work Package
API	Application Programming Interface

1 INTRODUCTION

1.1. Objective of this Document and Intended Audience

This document represents the deliverable 11 included within TASK-08: Design of the method for performing the code reviews for the European institutions.

The objective is to design a code review process to be used in the European Institutions, taking into account the results obtained on TASK-06: "*Requirement for the code reviews and their validity for the European Institutions"* and TASK-07: "*Analysis of the methods for communicating the results of code reviews, targeting their automated communication"*.

The design will be done based on a close collaboration with the European institutions including, but not limited to, document review and workshop validation.

1.2. Scope

To entirely understand the scope of the document, it is necessary to understand the aim of the Work Package (WP) 2. The WP2 has four tasks:

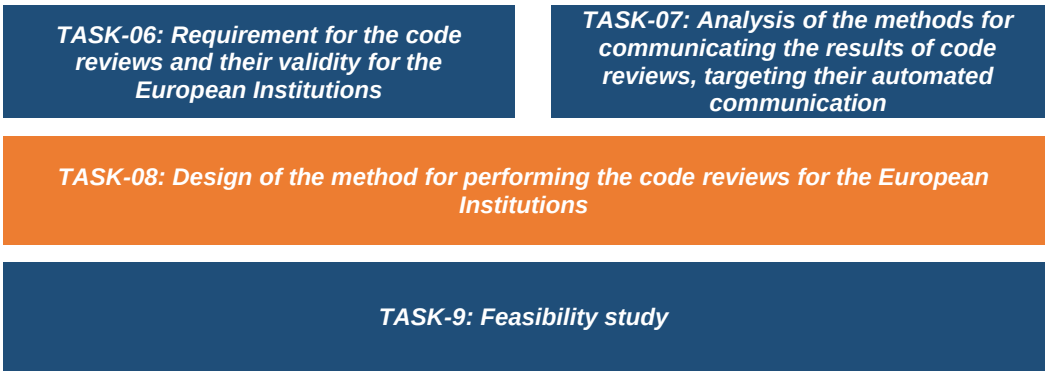
- Task 6: Requirements for the code reviews that aim to define the list of requirements for proper code reviews and their validity for the European Institutions, as well as to prepare an analysis of how they fit into the working methods of the European Commission and the European Parliament.
- Task 7: Analysis of the methods for communicating the results of the code reviews, targeting their automated communication.

  Tasks 6 and 7 will provide the requirements that the methodology defined in task 8 needs to fulfil. For this reason, deliverables 9 and 10 (output of tasks 6 and 7) are complementary.
- Task 8: Design of the code review process to be used in the European Institutions, taking into account the requirements defined in tasks 6 and 7.
- Task 9: Feasibility study of the method defined to perform code review, to be used in the European Institutions.

This is deliverable 11, the result of task 8, which covers the design and development of a code review methodology to be used to analyse open-source solutions provided by FOSS communities and European Institutions.

The selection of the tools that will be used for the code review and communication of the results have been selected on previous tasks (Task-06 and Task-07).

Figure 1: WP2 Tasks



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: **Approach**, which describes the tools selected for the pilot, based on the requirements established during Task-06 and Task-07.
- Section 3: **Methodology**, which defines the methodology to be followed on the code review processes.
- Section 4: **References**.
- Section 5: **Annexes**.

1.4. Key Success Factors

The following factors are needed to ensure the success of this phase:

- Selection of an appropriate tool, or set of tools, that adequately cover the requisites set on Task-06 and Task-07.
- Definition of a detailed code review methodology that ensures that all the requisites are covered and that it can be replicated easily by different teams or users.
- Definition of a robust validation process to ensure that any selection, methodology step or requirement is reviewed and approved by the stakeholders in order to ensure that it covers their needs.

1.5. Deliverables

1 *Deliverable 9: List of requirements for code reviews*

2 *Deliverable 10: List of methods for communicating the results of code reviews*

2 APPROACH

The approach followed on this methodology is based on the selection of one (or several) of the code review tools filtered on Deliverable 9, as well as considering the methods for communicating the results of the code reviews based on the tools identified on Deliverable 10. Therefore, the tools selected to be used to define the steps to follow in the methodology will be fully compliant with the needs of the European Institutions.

2.1. Code Review Tools

The following tools have been selected to be used as part of the methodology that is defined on the following chapters. This choice has been made taking into account the results shown in Deliverable 9, and **is to be exclusively used on this pilot**. It is not to be considered as an official nor general selection made by the European Institutions.

Table 1: Code review tools

Tool	Languages	Coverage
VCG	JAVA, PHP	77%
Yasca	JAVA, PHP	69%
RIPS	PHP	58%
FindBugs	JAVA	54%

2.2. Communication Tools

In order to define the methods to be followed in order to ensure proper communication and dissemination of the results of the code reviews, several additional, specific tools will also be selected and used.

These selections will be done taking into consideration the results obtained on Deliverable 10. As with the previous case, the tools are selected **exclusively for their use on this pilot**. They are not to be considered as an official nor general selection made by the European Institutions.

Table 2: Communication tools

Tool	Characteristics
JIRA	Used to distribute final report documents.
JoinUp	Used to distribute final report documents.
JSReports	Used to distribute the checks.

2.3. Operational Staff

In order to carry out these code reviews, the team that will be in charge will have to cover the following skills in order to better execute the phases, tasks and activities that have been proposed on the methodology and therefore present better results.

There will be several participants involved on a code review, each one having a specific set of responsibilities, and their participation being related directly to very specific tasks. The groups considered are:

Table 3: Participants

Participant	Responsibilities
Stakeholders	• Define the scope of the code review. • Validate final reports. • Manage and support the post-audit phase.
Code review team	• Define the tests to be carried out. • Configure and validate the environment to carry out the tests. • Carry out the execution of the test cases. • Generate the technical report including the results. • Develop and evaluate the impact analysis of the findings. • Provide an initial prioritisation and action plan. • Provide post-audit support.
IT team	• Provide high-level detail of the application to audit. • Provide the source code (if developed/maintained by them).
Developers	• Provide high-level detail of the application to audit. • Provide the source code (if developed/maintained by them).

On the other hand, within these participant groups, there will be a set of roles that have been defined and that have clearly listed their required skills and capabilities:

- Deep understanding of programming language, scripting language and other technologies used in the application.
- Good Knowledge of the latest testing tools.
- Good understanding of HTTP communication if they are testing a web application.
- Knowledge of basic vulnerabilities.
- Good reporting skills.
- Teamwork, communication and documentation skills.

3 METHODOLOGY

This chapter will contain a detailed definition of the phases to be followed during a code review analysis, the tasks to be carried out, and the procedure steps to achieve the objectives of these activities.

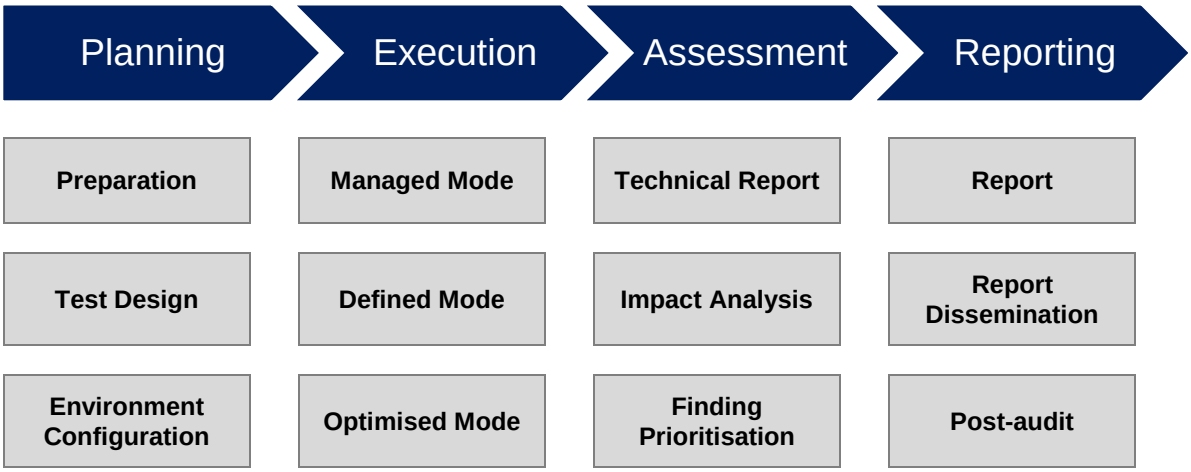
It has been divided into four sections:

- The first section will cover the code review methodology itself, defining all the phases, tasks and activities that have to be carried out to finish the code review successfully.
- The second section will detail the test cases (controls) that are defined for each one of the categories that are to be reviewed during the Execution Phase.
- The third section will detail the process to follow when a critical vulnerability is found on a third-party solution, either being part of the code being reviewed, or being a support solution.
- The final section describes the reporting process of the methodology in greater detail, explaining the steps to follow in order to communicate the results to the interested parties and generate the appropriate reports.

3.1. Code Review Methodology

The code review methodology that is developed in this deliverable is summarized on the following structure (see Figure 2), which covers all the phases from the initial definition of the code review process and up to the post-audit support whenever required. Each one of these phases will be divided into several main tasks that will summarize the main objectives of each phase, and which allows a better sorting of the activities that have to be carried out by all the participants that will be involved in the project.

Figure 2: Methodology phases

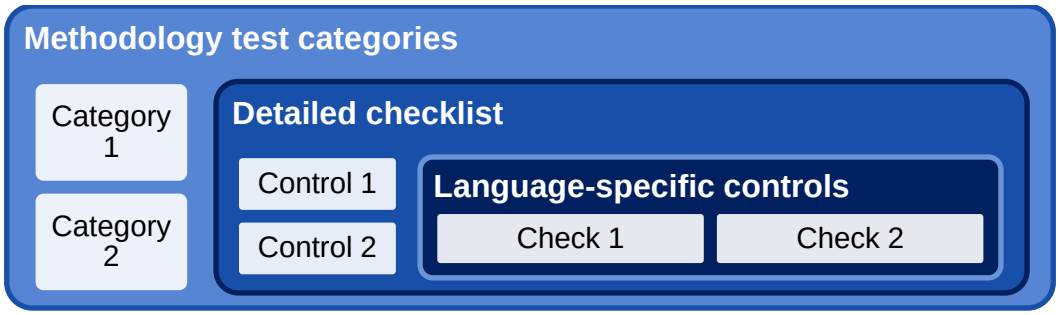


For each one of these phases, a summary table containing the main description of their objectives, tasks, expected results, constraints and dependencies is included. Now for each of the tasks defined for the phases, a second table is defined to include its own specific objectives and expected results, followed by a detailed description of the steps to be carried out by the code reviewers.

The estimated workload can be considered “static” for most phases and activities, with the exception of the Execution ones, which will vary depending on the size of the code and the controls to review. This is clearly mentioned in the corresponding activity detail tables.

On the execution phase, a set of controls will be verified by the code reviewers in order to properly analyse the code and its security. The Checklist in “Annex 1” contains a detailed list of the controls to evaluate within the categories described on this methodology (Section 3.24).

Figure 3: Test category control levels



As there are some controls that are language-specific, a third level of detailed controls is defined to include them within a supplementary checklist, and which will be carried out within the controls included on the detailed checklists (see Figure 3).

3.1.1. Planning

This stage will cover the information gathering activities that are required to start a code review analysis. This is a necessary step in order to ensure that the proper tests and tools are selected, and that the procedures can be properly optimised.

Therefore, initial basic information gathering activities will be carried out, as well as interviews with relevant stakeholders that requested the code review. This in order to define the scope, objectives and constraints of the project. Further on, a list of applicable test cases will be selected, taking into account the scope and constraints defined beforehand, and the testing environment will be prepared to make sure that all selected test cases can be carried out.

The following table consolidates the concepts that have to be covered in this phase:

Table 4: Methodology – 1. Planning phase details

Name	Code review planning phase
Objectives	
• Define the scope of the code review. • Establish the specific needs of the source code. • Obtain the necessary sponsorship within DIGIT. • Select the applicable tests that will be carried out based on the language and the individual needs. • Ensure that the selected tests cover all required security guidelines in use on the European Institution that requested the code review. • Prepare and configure the environment to allow the smooth execution of the tests that have been selected for this specific source code.	
Tasks	
Preparation	Define the objectives, time limit, scope and any specific needs or requirements of the source code.
Test design	Define the planning of the code review: tools to be used, applicable tests, execution times, effort required, etc.
Environment configuration	Prepare the environment and configure the tool in order to carry out the code review analysis.
Expected results	
- Agreement of the scope of the code review. - Sponsorship by DIGIT. - Distribution list (internal, restricted or public). - Definition of the specific needs and requirements of the source code. - Lists of applicable tests. - Planning of the code review considering the applicable tests. - Time and effort required to carry out the tests. - Environment configured and ready to be used for the code review.	

Dependencies	**Constraints**	**Other considerations**
There are no dependencies as this is the first phase.	1. Need sponsorship within DIGIT. 2. The complete source code to be reviewed must be available. 3. Restrictions defined by the stakeholders.	N/A

3.1.1.1. Preparation

The first task that has to be carried out to prepare the code review includes the definition of the scope and specific objectives for it. At this point, any constraints (time, resources, code availability…) will be identified to be taken into account into later stages.

The participation of the stakeholders that requested the code review is fundamental; however the IT teams' or open-source developer communities' assistance at this point will be optional, depending on the type of code to review.

Table 5: Preparation details

Name	Code review preparation	**Estimated workload**
Responsible	Code review project leader	3 days
Participants	Stakeholders (requesters of the code review) IT team in charge of the solution Open-source developers' communities	
Tools	N/A	

Validation	**Expected results**	**Exceptions**
Done by stakeholders before proceeding to the next phase.	- Scope defined. - DIGIT sponsorship obtained. - Specific needs identified. - High-level time estimation.	The participation of the IT team and communities is optional.

Steps to be carried out

The following activities will be carried out during this task by the different parties involved:

- Define the scope of the code review project.
  - Timeframe and delivery dates.
  - Code to review (name or identifier).
  - Amount of code (Nº lines, at least an approximate value).
- Establish the owner (stakeholder) of the code review request, which will be informed of any changes and will participate on all the follow-up and validation sessions.
  - Stakeholder, or stakeholders, in charge or the project.
  - Contacts of the IT team, communities or developers of the solution.
- Procure DIGIT sponsorship for the code review.
  - Establish a stakeholder responsible for all the logistics and communication coordination activities related to the code review.
- Establish the distribution list and communication channels:
  - People to include in the communications.
  - Visibility of the results:
    - *Internal*: DIGIT and EU Institutions only.
    - *Restricted*: DIGIT, EU Institutions and FOSS communities.
    - *Public*: available to the general public.
- Define the times for the developers to fix the issues found, and what steps to take if there are not fixed, or the developers refuse to take care of them.
- Define if critical findings identified during the assessment phase are to be communicated earlier:
  - If they are critical and have not been reported earlier (ex. in CVEs), it is recommended to communicate them immediately to all parties specified in the scope (will require DIGIT contact stakeholder approval).

  - If they are critical but are already known, they can be reported internally before the final report is provided if DIGIT indicates it so.
  - Under DIGIT's decision, it is also possible to not provide any critical communications before the final report is provided.
- Define the code review basic characteristics, including:
  - Programming language of the code.
  - Amount of code to be reviewed (Nº lines, exact amount).
  - Objective of the review (security check, bug/vulnerability detection…).
- Define the advanced characteristics of the source code to review.
  - Establish the applicable Defined Verification Level. Possible options are:
    - Level 1 - *Opportunistic*
    - Level 2 - *Standard*
    - Level 3 - *Advanced*
  - Architecture (high-level functional design) of the application to review.
  - Third-party libraries implemented within the code.
- Define any specific requirements that need to be covered during the code review by analysing the structure, dependencies and characteristics of the solution.
  - Third-party external services/solutions required.
  - Interfaces, plugins or extensions to include.
  - Specific configuration parameters.
- Generate an initial time and effort estimation, at a high-level.
  - Working days required.
  - Personnel required.
  - Specific skills or capabilities required.
  - High-level code review execution plan.

3.1.1.2. Test Design

Before starting this task, it is necessary to properly define the scope, objectives and constraints, as otherwise the test case selection will be inefficient and incomplete. Also, before reviewing applicable tests, the applicable tool has to be selected if there is more than one possibility.

Therefore, applicable test cases will be selected from the full checklists that are available for the code reviews (general checklist and specific language checklists), taking into account the scope and objectives that have been established.

If there are constraints defined that may affect the tests (time, resources, hardware…), the applicable test cases will have to be prioritised in order to select those that are most relevant for the code review.

Table 6: Test design details

Name	Code review test design		**Estimated workload**
Responsible	Code review team		1 day
Participants	N/A		
Tools	N/A		
Validation	**Expected results**		**Exceptions**
Revised by the code review project leader.	- Solution to be used. - List of test cases to run. - Configuration of the tests - List of discarded tests, including justification.		Ignore tests that do not apply. Select only a part of the applicable tests depending on the constraints.

Steps to be carried out

The following activities will be carried out during this task:

- Identify the code review solution to be used considering the language of the source code to be reviewed, specific requirements set during the previous phase, and the scope, objectives and constraints of the project.
    - Selection of the tool, or tools needed.
    - Plugins/extensions that provide needed features for the code review.
- Select the requirements that will have to be carried out in order to comply with the scope and objectives of the code review.
    - Generate a list of applicable test cases from all available ones (general and language-specific checklists).
    - Selection of those cases that cover the scope and the Defined Verification Level established during the previous task.
    - If there are Internal Security Guidelines in effect, it will be necessary to include test cases to cover their requirements.
    - List of ignored test cases, including justification for their exclusion.
- Establish a planning of the activities to carry out, including a time and effort estimation of each one of them.
    - Detailed execution plan including the tests to be carried out, including the list, order and expected time required to carry out the tests defined on this part, as well as the resources (users) needed for this.

3.1.1.3. Environment Configuration

The final task of this phase covers the need to prepare the environment where the code review will be carried out. This includes the need to install, configure and prepare all the tools that are going to be used, and ensure that any specific, non-standard libraries, services or extensions are readily available for the code review.

It is recommended to have these tools fully configured before starting the code review to avoid unexpected delays or issues later on due to the tools not being ready or not covering specific needs that had been previously identified.

Table 7: Environment configuration details

Name	Code review environment configuration	**Estimated workload**
Responsible	Code review team	1 day
Participants	N/A	
Tools	FindBugs * RIPS * VCG * Yasca *	

Validation	**Expected results**	**Exceptions**
Revised by the code review project leader.	- Environment configured for the code review.	N/A

** Only those selected on the previous phase.*

Steps to be carried out

The following activities will be carried out during this task:

- Determine the environment and solution needs taking into consideration the details of the project and the selected test cases:
  - Identify any hardware or software needs specific to this code review and install them accordingly.
  - Ensure that the environment is properly configured to protect the privacy of the code review process and of the code reviewed.
- Install/deploy the software tools to be used:
  - Deploy the tools on the prepared environment.
  - Configure the tool taking into account the specific needs of this code review.
- Install/apply any needed extension or plug-in:
  - Install and configure the plugins and extensions needed to cover the selected test cases.
  - Configure the plugins to allow their proper execution.

3.1.2. Execution

This stage will cover the execution of all the test cases that have been included for this code review, considering the scope, objectives and constraints. At this point, the execution will be done by the selected team, carrying both automated and manual test where necessary.

The following table consolidates the concepts that have to be covered in this phase:

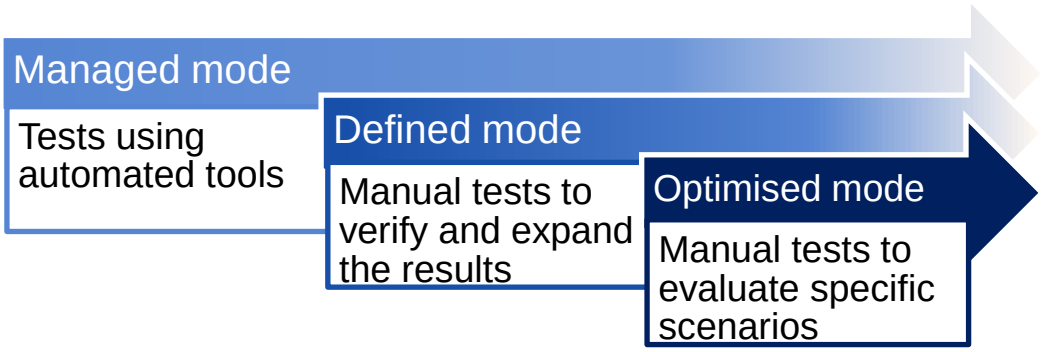
Table 8: Methodology – 2. Execution phase details

<table>
<tr><th>Name</th><td colspan="3">Code review execution phase</td></tr>
<tr><th colspan="4">Objectives</th></tr>
<tr><td colspan="4">• Execute the selection of test cases applicable for this code review, including both automated and manual tests.
• Identify weak points or sections that require further testing.
• Carry out optimised tests to evaluate weak points discovered or hard-to-detect vulnerabilities.
• Collect the execution results of these test cases.</td></tr>
<tr><th colspan="4">Tasks</th></tr>
<tr><td>Managed mode</td><td colspan="3">Execution of the test cases using automated tools that will provide initial results and known weaknesses/vulnerabilities, as well as serving as a starting point for the other more manual-oriented tasks.</td></tr>
<tr><td>Defined mode</td><td colspan="3">Comprises all tests that will be carried out also manually due to their nature, or those that require active supervision by a technician.</td></tr>
<tr><td>Optimised mode</td><td colspan="3">Includes all test conducted in Managed mode but focused to the software architecture and unique peculiarities of the code.</td></tr>
<tr><th colspan="4">Expected results</th></tr>
<tr><td colspan="4">- List of results from the automated tools.
- List of identified vulnerabilities, bugs and security weaknesses.
- Summary of each issue identified.
- Technical detailed information on each one of the issues identified.</td></tr>
<tr><th>Dependencies</th><th>Constraints</th><th colspan="2">Other considerations</th></tr>
<tr><td>Requires a list of the test cases to be executed.
The environment needs to be ready for the code review.
The code to be reviewed needs to be fully available.</td><td>Team requirements need to be covered.
Time and resource requirements need to be taken into account.</td><td colspan="2">N/A</td></tr>
</table>

Therefore, the steps to carry out have been divided into three consecutive tasks, each one providing further information and scope refining, allowing for a more in-depth analysis of the most critical sections of the code, as well as allowing the review of the vulnerabilities and weaknesses found. For this reason, the initial task focuses on the use of automated tools, while the second and third tasks shift towards the manual review of the findings.

Additionally, any bugs, vulnerabilities or typos discovered by the code reviewer that are not related with security, nor within the scope of the project, will be included in the conclusions section, citing them in a list format in order to allow the developers to be aware of them, assess them and fix if they consider it appropriate.

Figure 4: Code review execution tasks order



As can be observed in Figure 4, each one of the tasks of this phase is used as a starting point from the next one, complementing the results with each new phase until the end of the execution task.

3.1.2.1. Managed Mode

Automated tools will be used on this task to review those selected test cases that can be carried out with them, having minimal user intervention. The code review team will be in charge of the supervision of the analysis to verify it is working correctly and ensure that it generates the results accordingly.

Table 9: Managed mode details

Name	Code review managed mode execution	**Estimated workload**
Responsible	Code review team	50,000 lines per day
Participants	N/A	
Tools	FindBugs * RIPS * VCG * Yasca *	
Validation	**Expected results**	**Exceptions**
Revised by the code review project leader.	- Reports of the automated tools including weaknesses, vulnerabilities and bugs.	N/A

** Only those selected on the previous phase*.

Steps to be carried out

The activities that will be carried out during this task are mostly automated, requiring only limited user intervention, and include both passive and active checks. It will be able to cover a partial amount of the test cases that have been defined:

- Data/Input Management.
- Authentication Controls.
- Session Management.
- Authorization Management.
- Cryptography.
- Error Handling/Information Leakage.
- Logging/Auditing.
- Secure Code Design.
- Specific JAVA and PHP Controls.

The test cases contained in this phase are described in Section 3.4, therefore the tools will have to be configured to carry out the tests that are included within the scope of the project. These tools will generate a report with the results automatically. This will serve as a starting point for the next tasks (defined and optimised modes).

3.1.2.2. Defined Mode

The next task will take as an input the objectives and scope of the code review, alongside the results from the previous task (Managed mode). All these controls are reviewed manually (some test may be done using automated tools), focusing on those sections of the source code that have been identified as possible weak spots in order to perform a deep-level code review.

Table 10: Defined mode details

Name	Code review defined mode execution	**Estimated workload**
Responsible	Code review team	0,5h per control
Participants	N/A	
Tools	Eclipse (or similar development environment)	
Validation	**Expected results**	**Exceptions**
Revised by the code review project leader.	- Detailed list of weaknesses, vulnerabilities and bugs. - Initial technical report of the findings. - Test options to configure the optimised mode execution.	N/A

Steps to be carried out

The activities that will be carried out during this task are mostly manual and include both passive and active checks. The objective is to review each one of the following categories (considering only those included on the scope) and validate the results provided by the automated tools, further analysing any points that require more detail.

The description and breakdown of each one of these categories can be found in Section 3.2.

- Data/Input Management.
- Authentication Controls.
- Session Management.
- Authorization Management.
- Cryptography.
- Error Handling/Information Leakage.
- Logging/Auditing.
- Secure Code Design.
- Specific JAVA and PHP Controls.

During this stage, it is necessary to start to write the final report where the results will be contained. This report will include the sections defined on Annex 3.

For each control tested, it is necessary to complete its corresponding control table of the report, which contains the description and test process steps. The fields to complete are:

- **Checks**: indicate if the checks have been successful (✓), have failed (**X**) or are not applicable (**NA**).
- **Results**: detail any findings or information that validate the results provided to the test steps of the control.
- **Evidences**: include any additional evidence deemed necessary to justify the results.

The *Threat*, *Vulnerability* and *Impact* values do not need to be completed yet, as this will be done on the assessment phase.

3.1.2.3. Optimised Mode

This task will take as input the results of the managed and defined modes, in order to establish those categories or areas that require further testing as they are deemed as at risk of containing more complex weaknesses, vulnerabilities or hard-to-find bugs.

Table 11: Optimised mode details

Name	Code review optimised mode execution	Estimated workload
Responsible	Code review team	1h per control
Participants	N/A	
Tools	Eclipse (or similar development environment)	

Validation	Expected results	Exceptions
Revised by the code review project leader.	- Detailed list of weaknesses, vulnerabilities and bugs. - Detailed technical report of the findings.	N/A

Steps to be carried out

The activities that will be carried out during this task are mostly manual and include both passive and active checks:

- Check those categories from the Defined Mode or Managed Mode which require manual validation or further evaluation.

Execute additional tests that are not covered on the previous categories, including:

- Concurrency (thread management).
- Evaluate possible Denial-of-Service (DoS) and Distributed Denial-of-Service (DDoS) weaknesses.
- Evaluate the Memory management processes (files, buffers, connections, sensitive information contained in variables, etc.).
- Analyse the Resource management processes (resource release, access control, temporary resources' management, etc.).
- Analyse the management of multiple calls to the same process, API or internal function.
- Review the Role/Privilege matrix (determine if it has a proper design, structure and assignation – outside of the code itself).

The results from this task will also be included on the report, following the same steps as the ones indicated on the Defined Mode task.

3.1.3. Assessment

This phase covers the analysis and evaluation of the findings identified on the code reviewed via the managed, defined and optimised modes. This will require the evaluation of the technical report and the details of each weakness, vulnerability or bug found, as well as the execution of an impact analysis activity to define the risk posed by each finding. Finally, taking into account the results of the impact analysis, the findings will be prioritised in order to easily identify those that should be fixed first.

The following table consolidates the concepts that will be covered in this phase:

Table 12: Methodology – 3. Assessment phase details

<table>
<tr><th>Name</th><td colspan="3">Source code review results' assessment</td></tr>
<tr><th colspan="4">Objectives</th></tr>
<tr><td colspan="4">• Carry out an evaluation of the issues found on the previous stage (Managed, defined and optimised modes) in order to identify their properties and their details.
• Carry out an impact analysis that will allow the definition of the risk posed by each one of the findings.
• Prioritise the findings in order to highlight those that pose a more critical risk for the code solution and provide a recommended fix order for all the findings.</td></tr>
<tr><th colspan="4">Tasks</th></tr>
<tr><td>Technical report analysis</td><td colspan="3">Analysis of the results from the previous phase (contained on the detailed technical report) in order to determine their characteristics and possible solutions. At this point, these initial results could be shared with the communities to promote their own evaluations.</td></tr>
<tr><td>Impact analysis</td><td colspan="3">Evaluation of the findings of the code review in order to determine their Threat, Vulnerability and Impact. At this point, a custom variant of a standardised score methodology, such as CVSS (v3) or CWSS will be used.</td></tr>
<tr><td>Finding prioritisation</td><td colspan="3">Classify the findings found based on the results of the previous impact analysis, considering the risk they pose to the solution reviewed. Provide an action plan sorting the findings and their solutions based on this classification.</td></tr>
<tr><th colspan="4">Expected results</th></tr>
<tr><td colspan="4">- Technical detailed report of the vulnerabilities, bugs and possible security weaknesses found.
- Analysis report of the findings and issues found on the code review.
- Evaluation and scoring of the findings based on their Threat, Vulnerability and Impact.
- Action plan including a prioritisation of the findings.</td></tr>
<tr><th>Dependencies</th><th>Constraints</th><th colspan="2">Other considerations</th></tr>
<tr><td>Results from the execution phase, containing detailed information of each issue found.</td><td>Execution reports from the previous phase, including the technical detailed report.</td><td colspan="2">N/A</td></tr>
</table>

In order to evaluate the impact of the findings of the code review analysis a series of indicators will have to be taken into account. It is important to consider that these indicators will help to determine the final scoring of the vulnerabilities/weaknesses.

- **Threat**: focuses on those factors that are directly related with the attack vector, and more specifically the probability that the attacker (either intentionally or unintentionally) manages to successfully take advantage of the issue.
- **Vulnerability**: those details specifically related to the vulnerability itself, focusing on the chance of its discovery and/or exploitation.
- **Impact**: it is centred on the common security concepts (confidentiality, integrity and availability) and how the issue identified affects them.

If the vulnerability has not been discovered yet, it can be considered for submission to be scored as a CVSS or CWSS depending on its impact and effect.

3.1.3.1. Technical Report Analysis

The first task focuses on the analysis of the detailed technical report generated on the previous phase, in order to determine the specific vulnerabilities found, their details and their possible mitigation or solution actions that should be followed. In this task, the findings will be classified based exclusively on their category.

Table 13: Technical report analysis details

Name	Technical report analysis	**Estimated workload**
Responsible	Code review team	3 days
Participants	N/A	
Tools	N/A	

Validation	**Expected results**	**Exceptions**
Revised by the code review project leader.	- Detailed review of the findings of the execution. - Classification of the findings based on their category.	N/A

Steps to be carried out

The following activities will be carried out during this task:

- Analyse and review the findings of the code review contained on the initial technical report generated on the previous phase:
  - Discard any false positives.
  - Assess the relevance of the findings.
  - Assign risk scores based on the confidentiality, integrity and availability of the data managed by the application.
  - Group the findings into their corresponding controls.
  - Sort the controls and findings into their respective categories.
- Review the content of each control and finding and complete any missing information, including but not limited to description, characteristics, technical details and evidences.
- Evaluate and generate possible solutions for the findings in each one of the controls, at high level:
  - General recommendations.
  - Mitigation actions.
  - Solutions and fixes.
- Transfer the findings, once they are grouped and sorted, into the checklists, filling all the fields with the exception of the risk.

3.1.3.2. Impact Analysis

Once the findings have been properly documented and classified (based on their category), the next task covers the need to determine their Threat, Vulnerability and Impact. This analysis is done from the point of view of the needs and structure of the client. Industry standards will be used to determine the vulnerability level of the risks found on the code review.

Table 14: Impact analysis details

Name	Impact analysis	**Estimated workload**
Responsible	Code review team	2 days
Participants	N/A	
Tools	N/A	

Validation	**Expected results**	**Exceptions**
Revised by the code review project leader.	- Detailed impact analysis of each one of the findings including scoring. - Classification of the findings based on their score.	N/A

Steps to be carried out

The following activities will be carried out during this task:

- Evaluate the sorted findings to determine the risk they pose.
  - From a high-level point of view, considering the controls as individual findings.
  - Refining the analysis of each control, reviewing each one of their occurrences.
- Determine the severity of the issue by calculating the risk value of the Threat, Vulnerability and Impact factors.

Table 15: Threat, Vulnerability and Impact possible values

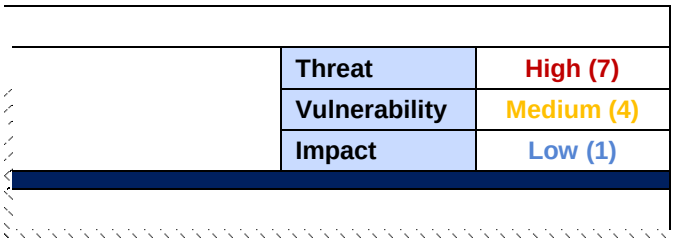
Factors		Values
Threat	*Skill required*	(1) Advanced intrusion and exploitation skills (2) Advanced computer coding and network skills (4) Advanced computing skills (6) Technical knowledge (9) No specific knowledge
	Opportunity	(1) Full access or expensive resources needed (5) Special access or specific resources needed (7) Limited access or standard resources needed (9) No access or resources needed
	Dimension	(2) Administrators or developers (4) Internal users/partners (6) Authenticated users (9) Anonymous users
Vulnerability	*Ease of discovery*	(1) Very hard (3) Hard (7) Easy (9) Discoverable with automated tools
	Ease of exploitation	(1) Theoretical (3) Hard (5) Easy (9) Exploitable with automated tools
	Awareness	(1) Unknown (4) Hidden (7) Obvious (9) Publicly known
Impact	*Confidentiality*	(2) Low amount of non-sensitive data leaked (6) Low amount of critical data leaked (6) Large amount of non-sensitive data leaked (7) Large amount of critical data compromised (9) All data (non-sensitive and critical) leaked
	Integrity	(1) Low amount of data partially corrupted (3) Low amount of data severely corrupted (5) Large amount of data partially corrupted (7) Large amount of data severely corrupted (9) All data severely corrupted
	Availability	(1) Few secondary services/functions affected (5) Few primary services/functions affected (5) Multiple secondary services/functions affected (7) Multiple primary services/functions affected (9) All services/functions affected

- Evaluation of the scores obtained for the Threat, Vulnerability and Impact of each issue, considering the following scale conversion:

  0 to 3: Low **3 to 6:** Medium **6 to 9:** High

  These results will be included on the detailed control tables that are part of the final report.

Figure 5: Detailed control risk results (sample)



- Finally, the checklist will be completed adding the **global risk** posed by the controls, which is usually calculated from the individual results (Threat, Vulnerability and Impact). Table 16 shows how to calculate the global risk taking into consideration the Impact and the probability (Average value of both Threat and Vulnerability results).

Table 16: Global risk evaluation

	High	Medium	High	Critical
Impact	**Medium**	Low	Medium	High
	Low	Info	Low	Medium
		Low	**Medium**	**High**

Probability (Avg. Threat & Vulnerability)

The possible values are ***Critical***, ***High***, ***Medium***, ***Low*** or ***Info*** if an issue has been found (**X**); or *empty* if no issues were found (✓), or the control is not applicable (NA).

Figure 6: Checklist control risk results (sample)

ID	Control	Result	
<id>	<control_name>	X	Critical

- At this point, there is a particular situation that can occur if a critical and unknown vulnerability is found, as it is of the highest interest to inform all relevant parties involved in order to ensure that it can be fixed in a timely manner.

  This would require the code reviewer to inform the stakeholder in charge of the finding, and communicate this finding via specific channels; as long as this has been agreed upon on the planning phase.

  Three possible scenarios appear:

  - If the solution is being developed, or maintained, internally, then it is only necessary to inform the relevant stakeholders and provide them with the vulnerability information before continuing with the assessment.
  - If the solution is being distributed by an external party, and within the scope of the project, the relevant developers will be provided with the information of the vulnerability as long as this communication is approved by the stakeholders in charge of the code review.
  - If the solution is being distributed by an external party, that is not within scope of the project, then the decision on whether providing the information early (before the final report) to these developers will fall on the stakeholders in charge of the code review.

3.1.3.3. Finding Prioritisation

Finally, once the impact analysis is finalised, and all the findings are scored taking into consideration their Threat, Vulnerability and Impact scores, an action plan is then designed to establish the recommended order in which they should be fixed.

Table 17: Finding prioritisation details

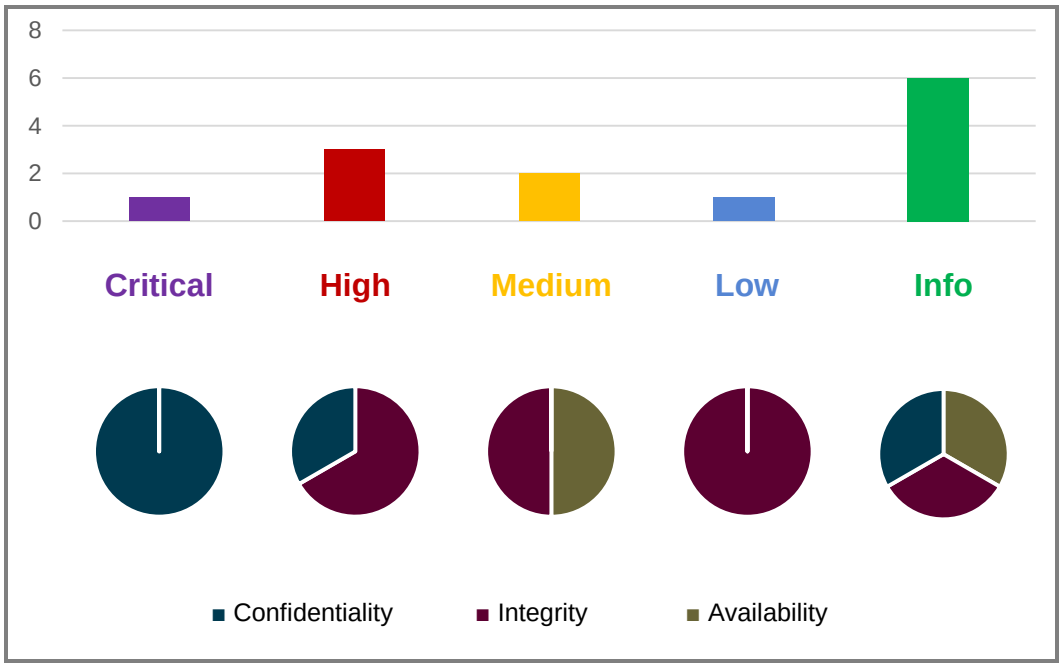
Name	Finding prioritisation	**Estimated workload**
Responsible	Code review team	2 days
Participants	N/A	
Tools	N/A	
Validation	**Expected results**	**Exceptions**
Revised by the code review project leader.	- Prioritisation of the findings based on their impact score. - Action plan detailing the order in which to address the findings.	N/A

Steps to be carried out

The following activities will be carried out during this task:

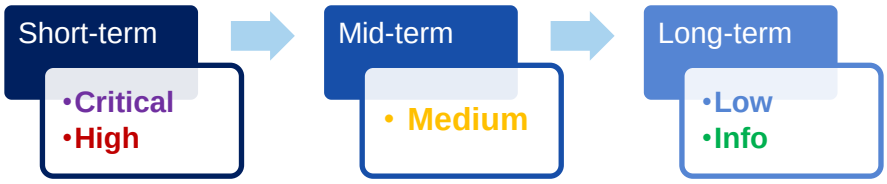
- Grouping of the findings found based on their global value, including breakdowns of their separate Threat, Vulnerability and Impact scores. These results are displayed on an executive indicator report that summarizes the result:
    - Bar graph showing the number of findings and their severity.
    - For each severity level, a pie chart showing the breakdown of the impact (Confidentiality, Integrity and Availability) risk level values.

Figure 7: Executive report finding indicators



- Prioritisation of the findings in order to define an action plan, taking into consideration the scoring results of each finding.
    - Classification of the findings based on their global risk score.
    - Classification of the findings based on their category and the Threat, Impact and Vulnerability results.
- Define an action plan that includes the steps and actions needed to mitigate and/or fix the findings.
    - Main action plan considering all findings.

Figure 8: Priority levels (sample)



- If required by the stakeholder, and it has been included and foreseen in the code review planning phase, it is also possible to include individual action plans per category, module affected or similar.

3.1.4. Reporting

This final phase covers the reporting of the results to the stakeholders, providing the conclusions, recommendations and action plans defined on the previous phases. Furthermore, an optional task has been included to carry out post-audit verifications to ensure that the findings have been mitigated and/or solved properly.

The following table consolidates the concepts to cover in this phase:

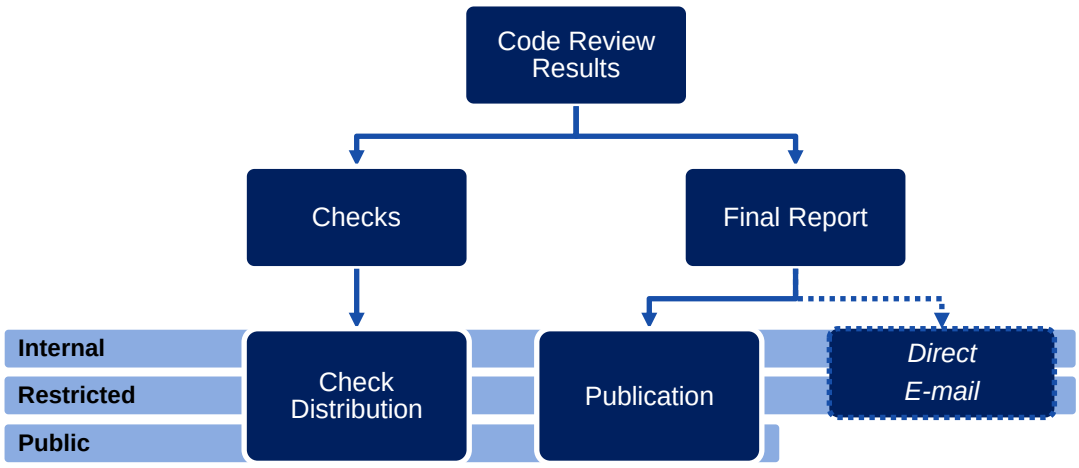
Table 18: Methodology – 4. Reporting phase details

Name	Source code review reporting
Objectives	
• Communicate the results to all parties included within the scope of this code review, as well as any communities or developers that are considered relevant if critical vulnerabilities are found. • Develop the final executive and detailed technical reports taking into consideration the results, recommendations and conclusions obtained on the previous phases of the code review. • Validate these reports with the stakeholders in charge of the reports to ensure that they comply with the scope and objectives set for the analysis. • Provide post-audit support for the implementation and verification of the solutions and mitigation actions recommended on the final reports to fix the findings identified.	
Tasks	
Report	Develop the final executive summary and final detailed technical report including all the results, recommendations and conclusions of the code review.
Report dissemination	Disseminate the final reports to the selected recipients (as indicated on the scope of the project). If applicable, communicate the results to external communities, third-parties or the general public.
Post-audit support	Provide support during the implementation of the recommendations, solutions and mitigation actions proposed on the final report.
Expected results	
- Final executive summary. - Final detailed technical report. - Disseminated communications to the parties in scope. - Action plan to manage the post-audit solution/mitigation implementation support.	

Dependencies	Constraints	Other considerations
Assessment results, finding prioritisation action plan and detailed recommendations. Detailed technical report including all the findings obtained during the Execution phase.	N/A	N/A

The following diagram (Figure 9) represents the different communication channels that are considered for the distribution of the reports and checks, in accordance with the requirements set on Deliverable 10 and detailed further on in the next sub-sections.

Figure 9: Possible reporting channels



As it can be observed on the diagram, the results of the code review will be reported in two main documentation groups: final report documents and checks. At the same time, the final report documents will be published using the selected tool and, if defined on the scope, sent via e-mail to relevant stakeholders or key personnel.

3.1.4.1. Report

The main focus of this task is to develop the final reports that will be handled as a result of the code review carried out. Two reports will be generated; one from a high-level point of view, containing an executive summary and an overview of the findings, planning and recommendations, and a second one containing detailed descriptions and recommendations for each one of the findings identified.

Table 19: Reporting details

Name	Code review reporting		**Estimated workload**
Responsible	Code review team		3 days
Participants	Stakeholders		
Tools	N/A		
Validation	**Expected results**		**Exceptions**
Revised by the code review project leader. Reviewed and validated by the stakeholders.	- Final executive summary. - Final detailed technical report.		N/A

Steps to be carried out

The following activities will be carried out during this task:

- Distribute the findings of the code review process using the tools indicated on Section 2.2.
- Develop the reports to be shared with the stakeholders as a conclusion of the code review:
    - High-level executive summary, following the structure provided in Annex 3 of this document, including a high-level summary and an overview of the code review findings.
    - Detailed technical report, following the structure provided in Annex 3 of this document. It has to include all the results obtained on the code review, including the results, mitigation actions and fixes.
- Develop a set of specific recommendations related to the findings and mitigation actions/fixes.
- Include the action plan defined on the previous phase, taking into consideration the recommendations and any changes that should be considered.
- Generate the final documents and send them to the stakeholders for validation.
- Review the stakeholders' comments and update the document where necessary.
- Distribute the final report to the stakeholders and any other community or third party identified on the start of the code review project.

3.1.4.2. Report Dissemination

Once the final reports are validated, they will be distributed among the agreed recipients, which can include internal staff, communities or general public. By default, both the final report documents and the results are distributed. The means of distribution have already been described on Figure 9, and may include sending the final report documents via e-mail to relevant or critical stakeholders and key personnel.

Table 20: Report dissemination details

Name	Report dissemination	Estimated workload
Responsible	Stakeholders	2 days
Participants	N/A	
Tools	JIRA / JoinUp (final report documents) JSReports (checks)	

Validation	Expected results	Exceptions
N/A	- Results and reports communicated to all agreed third parties/communities.	N/A

Steps to be carried out

The following activities will be carried out during this task:

- Distribution of the final report documents to all parties included within the scope of the code review:
  - Publication of the document on the tool selected, and distribution of the access link or awareness communications to inform the parties of the publication.
  - Sending the documents via e-mail to relevant or critical stakeholders and key personnel, but only for cases previously agreed during the planning stage of the code review.
- Communication of the results from the code review findings, using the tool selected on the previous deliverable.
  - Inform the parties within scope of the publication of the results.
  - Verify that intended recipients have access to the platform with the published checks.
  - For cases where public access has been included within scope, consider including an external front-end to allow information visualization to facilitate the access to the information.

3.1.4.3. Post-audit Support

This optional task is designed to provide support for the verification of the correct implementation of the recommendations proposed on the final report, ensuring that the issues and findings are properly fixed. This requires a faster, to-the-point code review.

Table 21: Post-audit support details

Name	Code review post-audit support		**Estimated workload**
Responsible	Code review team		# required days
Participants	Stakeholders		
Tools	N/A		
Validation	**Expected results**		**Exceptions**
Revised by the code review project leader. Reviewed and validated by the stakeholders.	- Final executive summary. - Final detailed technical report.		N/A

Steps to be carried out

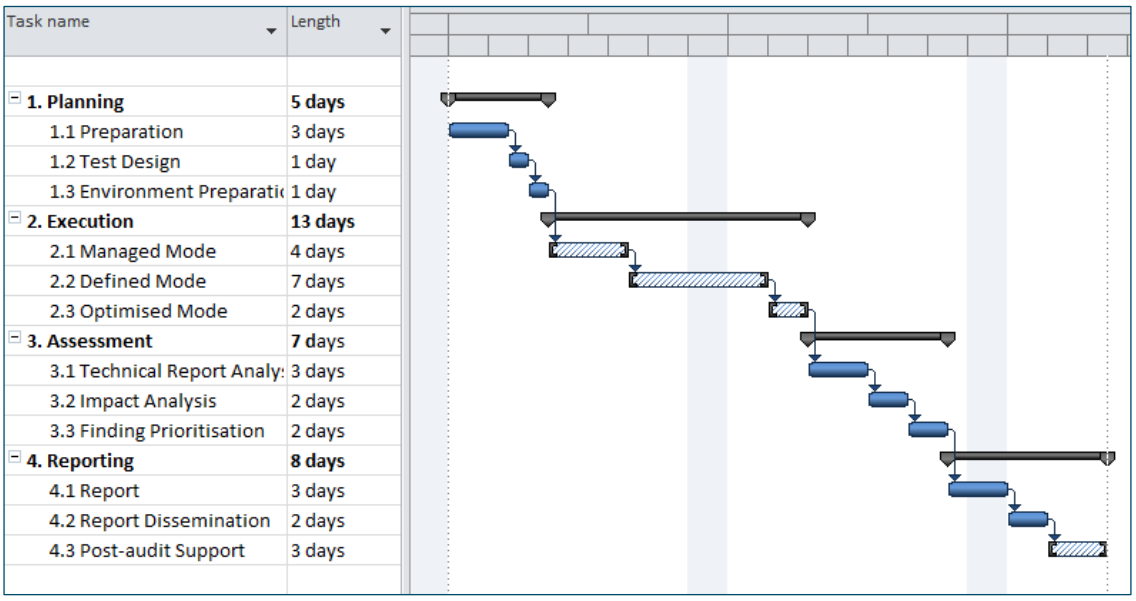
The following activities will be carried out during this task:

- Participate in any follow-up activities designed and carried out in order to solve the findings of the code review:
  - Recommendation implementation support, both during the design, coding and implementation phases.
  - Establishment of deadlines for solving the findings (developers).
  - If developers are unwilling/unavailable, the stakeholders will have to select who is going to carry out and distribute the fixes.
  - Validation of the correct implementation of the recommendations.
- Verification of the correct solution of the findings:
  - Carry out the corresponding controls and tests to ensure that the findings have been fixed correctly.

3.2. Project effort planning

The time to carry out the project will vary depending on the code to be reviewed and the controls selected for the code review. A sample effort planning can be seen in Figure 10; for this sample maximum times have been considered (all controls tested for general and language specific categories).

Figure 10: Project effort planning (sample)



3.3. Responsibilities and assignations

The roles defined on Section 2.3 do not participate in each phase. Instead, their participation is focused on those sections where they can provide support, information or needed data. Table 22 contains a list of the main phases, and their sub-phases, of the methodology, determining the task each participant has to carry out.

Table 22: RACI matrix

PHASE	DAYS	PARTICIPANTS			
		Code review team	Stakeholders	IT team	Developers
Planning	**5**				
Preparation	3	R	A	C	C
Test Design	1	AR	I	C	C
Environment Configuration	1	AR	I	--	--
Execution	**13***				
Managed Mode	4*	AR	--	--	--
Defined Mode	7*	AR	--	--	--
Optimised Mode	2*	AR	--	--	--
Assessment	**7**				
Technical Report	3	AR	--	--	--
Impact Analysis	2	AR	CI	I	I
Finding Prioritisation	2	AR	--	--	--
Reporting	**5**				
Report	3	R	A	--	--
Report Dissemination	2	R	AR	I	I
Post-audit support	N/A	R	AR	C	C

* Estimation considering all possible controls and checks.

Legend:

- **(R) Responsible**: actor is assigned to carry out the task.
- **(A) Accountable**: actors in charge that have the authority to take decisions about the tasks.
- **(C) Consulted**: actors that can provide additional data or information regarding the tasks.
- **(I) Informed**: actors that need to be informed of the progress/results of the phase.

3.4. Test Categories

The following groups of tests and areas to test are considered for its use on the Execution Phase. These groups include architectural and code review controls and checks, merged as they, in many cases, overlap each other. This merge allows for a more detailed and appropriate analysis. These categories are the following:

Data/Input management

The data entry points of an application, service or library are one of the weak points that must always be controlled against unexpected values to protect the integrity and security of the application, this includes:

- *Entry point identification*: all data entry points must be identified, regardless of the origin of the data (e.g. libraries, functions, user interaction…), and must be properly documented.
- *Entry point validation*: review that all data entry points have specific validation controls that filter invalid data, code fragments or function instructions (e.g. SQL injection).
- *XML schema validation*: if XML data entry points are identified, their schema has to be validated to ensure that it is robust, and there must be controls in place focused on their validation (structure, schema and content).
- *Data whitelisting/blacklisting*: if available, any whitelist or blacklist configurations in place must be checked to ensure that they are implemented properly and that they are used as expected (filtering untrusted sources, invalid data format…).

Authentication controls

It covers any aspect related to the process where the solution reviews and verifies the identity of another entity, such as a user, via the use of various authentication mechanisms (user credentials, biometrics, etc.).

- *Authentication process verification*: evaluation of the authentication methods used to grant access to authorized entities.
- *Password policy usage*: ensure that there is a password policy is in use to control the user credential strength, and determine the adequacy of this control verifying that it provides adequate protection.

- *Credential storage security*: evaluate how the user credentials are stored in the solution, if encryption or hashing is used to protect the password from unauthorised access.
- *User account protection*: verify that the user accounts are protected against unauthorised use, including lockout times, password expiration and connection location validation.
- *Password recovery process*: identify if a manual password recovery process is available and review the information required to validate the soliciting users' identity.
- *Two-factor authentication*: identify any two-factor systems provided by the applications for their users, identifying their robustness, optional features and use of different vectors (passwords, biometrics…).

Session management

Covers all aspects of the protection and management of user sessions once they are authenticated against the solution.

- *Session creation*: verify that the session creation process assigns the appropriate access and control privileges related to the user for which the session is being created.
- *Session ID management*: the ID assigned to the session must be unique and never be reused for the same or a different user. Similarly, the ID should not contain any personally identifiable data.
- *Session lifecycle*: the solution should be able to manage adequately the session throughout its lifecycle, starting when it is created, defining timeouts, inactivity checks and other restrictions imposed to protect it.
- *Session logout*: once the user exits the session, or the timeout is reached, the application must be able to properly finalize the session, closing any open connections and removing any temporarily assigned permissions.

Authorisation management

This process is designed to ensure that when a user or entity correctly authenticates against the application, it gets the proper privileges assigned to it. These privileges should be assigned based on the role and any specific permission that the user has, and should update accordingly if there is a modification on them.

- *Components' implicit trust*: verify that any communications between different modules or components of the application make use of authentication methods and verify that the other components' credentials are valid.
- *Access control systems*: evaluate any access control system in use on the application, reviewing the control rules applied, the process followed and the exception control it makes use of (special cases, multiple errors, etc.) and the usage of predefined roles or privilege schemas. There are three main access control systems to review (if in use):

  - Role Based Access Control (RBAC).
  - Mandatory Access Control (MAC).
  - Discretionary Access Control (DAC).

- *Privilege revision*: if the solution has more than one role defined, then each role will have different permissions. Therefore, apart from the use of a Privilege Schema (previous check), the application should include specific controls designed to ensure that the privileges are assigned and enforced correctly.
- *Trust levels*: if the solution accepts connections from external entities, there is a need to establish different trust levels for the available entry points assigning only the must-have privileges, applying the Principle of Least Privilege (POLP).

Cryptography

Covers all aspects related to the protection via encryption of the information and data in transit and at rest.

- *Credential protection at rest*: verify that any credentials that are stored by the application have the appropriate measures for their confidentiality. The optional solution is to store a non-reversible hash of the password, encrypted for security. The minimum level of security is provided by simply encrypting the credentials and storing them locally.
- *Credential protection in transit*: review the security methods applied to protect credential information in transit. Whenever possible, hashes should be sent instead of the credentials, and these must be encrypted and protected to avoid interception and replay attacks.
- *Cryptographic libraries in use*: determine the cryptographic libraries used, if any, and obtain their version to identify any possible known vulnerabilities or risks.
- *Cryptographic configuration*: evaluate any cryptographic parameters used by the application and used to configure the libraries, algorithms or cyphers in use.

Error handling /Information leakage

The security of the internal information of the application is fundamental for its protection. The information provided by the applications' errors must be filtered to avoid providing any sensitive or internal information; on the same page, metadata, sample pages and tutorials must avoid mentioning any sensitive information.

- *Information leakage*: the code must be reviewed, especially web content sent to the client and scripts, in order to ensure that no sensitive information is leaked. On the other hand, debugging information, functions and reports must also be disabled on production environments.
- *Sample files*: ensure that there are no sample or backup files available on the application, or if any are provided, ensure that they contain no sensitive or internal information on them.
- *Error handling*: verify that errors generated by the application are fully controlled and do not contain any sensitive or internal information.

Logging /Auditing

The logs generated by an application are a superb source of information about its contents, workings and potential weaknesses. Therefore, these must be protected in order to avoid any unauthorised access or leakage of this information.

- *Log configuration management*: review the log configuration parameters to ensure that the proper information is gathered and stored.
- *Log generation*: review the log generation process used by the application, including the specific phases the log goes through.
- *Log storage*: verify that logs are stored securely and that they can only be accessed by the appropriate authorised users.
- *Log integration*: review the integration options provided by the log manager in relation with the log provision to other systems (syslog, backup systems…).
- *Log sensitive information*: verify that no sensitive information is included on the logs; if such information has to be provided, it will be masked or hashed to avoid identification.

Secure code design

There are several aspects related to the application itself and the technologies and frameworks used for its implementation.

- *Framework implementation*: if any frameworks are used on the application, they should be identified, listed and their implementation verified to ensure that it does not put the application, the data it contains or its users at risk.
- *Configuration parameters*: identify how the configuration of the application is being managed, including its storage, management, validation and application.
- *JavaScript usage*: the use of JavaScript on the client side must be controlled and only used in cases where it is strictly necessary. The methods used must be reviewed to make sure that no insecure or risky functions are used. Any validation done in the client-side via JavaScript must also be done again on the server side.
- *Variable types / operations*: the operations carried out within the code regarding the different variable types must be controlled to avoid unexpected results, effects or impact on the application.
- *Expressions / Methods*: functions and expressions must be controlled to avoid parsing null values, properly manage functions and ensure correct expression management and transformation.

Optimised mode controls

Taking as input the results from the managed and defined mode, this part will cover very specific cases that require further testing, as the automated/manual controls do not properly cover them.

- *Concurrency*: review the thread generation and management process, identifying all specific controls in use and determining their proper working.
- *Denial of Service (DoS)*: review weak points defined on the previous phases that could be vulnerable to denial of service attacks.
- *Memory and resource management*: evaluate how memory allocation is managed by the code, ensuring that is properly reserved when needed and liberated when it is no longer required.
- *Code structure*: evaluate all those processes in which the calls are made back into the code itself, making sure that these calls do not overwrite previous ones in progress. At the same time, it is fundamental to ensure that the application layers are properly and securely separated.
- *Role-privilege matrix*: review the privilege and role schemas in order to identify conflicts or permission issues.

4 REFERENCES

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

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

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

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

5 ANNEXES

The following annexes contain the checklists to be used during the execution of the methodology, including variants for each possible language to be analysed (such as JAVA or PHP), and the structure for the final reports.

5.1. Annex 1: Control Checklist

This annex contains a checklist of all the controls that can be reviewed on a given code review project. These controls have to be complemented with the corresponding checklist of the specific language of the code to be reviewed (ex. JAVA specific requirements checklist, also in this Annex). Detailed information about each one of the controls is included on Annex 2.

The checklist table is composed of four columns:

- Unique identifier (ID) of the control.
- Name of the control.
- Control applicability.
- Overall risk of the issues found for the control.

For each category, several subcategories have been defined, therefore, the following ID format has been applied:

#CAT# - #SUB# - #000#

#CAT# is a three-letter abbreviation of the category name.
#SUB# is a three-letter abbreviation of the subcategory name.
#000# is the number of the control.

5.1.1. Data/Input Management (DIM)

The controls in this category have been grouped into the following subcategories:

- File Input / Output Management (FIM)
- Data streams management (DSM)
- Character encoding management (CEM)
- Input validation and sanitization (IVS)
- Sensitive Data Management (DSM)

5.1.2. Authentication Controls (AUT)

The controls in this category have been grouped into the following subcategories:

- Authentication verification (id: AUV).
- Password policy usage (id: PPU).
- Credential storage security (id: CST).
- User account protection (id: UAP).
- Password recovery process (id: PRP).
- Two-factor authentication (id: TFA).

5.1.3. Session Management (SMG)

The controls in this category have been grouped into the following subcategories:

- Session creation (id: SCP).
- Session ID management (id: SID).
- Session lifecycle (id: SLC).
- Session logout (id: LGP).

5.1.4. Authorisation Management (ATS)

The controls in this category have been grouped into the following subcategories:

- Component implicit trust (id: CIT).
- Access control system (id: ACS).
- Privilege revision (id: PRV).
- Trust levels (id: TLV).

5.1.5. Cryptography (CPT)

The controls in this category have been grouped into the following subcategories:

- Credential protection at rest (id: CPR).
- Credential protection in transit (id: CPC).
- Cryptographic libraries in use (id: CLU).
- Cryptographic configuration (id: CRC).

5.1.6. Error Handling /Information Leakage (EHI)

The controls in this category have been grouped into the following subcategories:

- Information leakage (id: INL).
- Sample files (id: SFL).

- Error handling (id: EHD).

5.1.7. Software Communications

The controls in this category have been grouped into the following subcategories:

- HTTP Secure Management (HSM)
- Secure Socket transmissions (SST)
- Web Services (WBS)

5.1.8. Logging/Auditing (LOG)

The controls in this category have been grouped into the following subcategories:

- Log configuration management (id: CFG).
- Log generation (id: GEN).
- Log storage (id: STG).
- Log integration (id: INT).
- Log sensitive information (id: LSI).

5.1.9. Secure Code Design (SCD)

The controls in this category have been grouped into the following subcategories:

- Framework requirements (id: FWK).
- Configuration parameters (id: CFG).
- JavaScript usage (id: JSC).
- Variable types / operations (id: VTY).
- Expressions/Methods (EXM).

5.1.10. Optimised Mode Controls (OPT)

The controls in this category have been grouped into the following subcategories or main tasks. The approach of these tasks is different so there are less controls defined in favour of a more customized analysis depending on the scope, objectives and code to be reviewed:

- Concurrency (id: CCR).
- Denial of Service (id: DOS).
- Memory and resource management (id. MRM).
- Code Structure (id: COS).
- Role-privilege matrix (id: RPM).

5.1.11. Specific JAVA Control Checklist (J)*

A checklist that contains all the controls that can be reviewed on JAVA based code reviews. These results are complementary of the common checklist that has been defined on this document. The categories and subcategories that include new controls are:

- Data/Input Management (DIM): Entry point identification (id: EPI)
- Authentication controls (AUT): Authentication verification (id: AUV)
- Error handling/Information leakage (EHI): Error handling (id: EHD)
- Secure code design (SCD): Variable types / operations (id: VTY)
  Expressions / Methods (EXM)
  Java Platform security (id: JPS)
  Runtime environment (id: ENV)
  Serialization (id: SER)
- Optimised mode controls (OPT): Concurrency (id: CCR)
  Visibility and atomicity (id: VNA)

5.1.12. Specific PHP Control Checklist (P)*

A checklist that contains all the controls that can be reviewed on PHP based code reviews. These results are complementary of the common checklist that has been defined on this document.

- Data/Input Management (DIM): Entry point validation (id. EPV)
  File handling (id. FHD)
- Session management (SMG): Session creation (id. SCP)
  Session lifecycle (id. SLC)
- Authorization management (ATZ): Privilege revision (id .PRV)
- Error handling/Information leakage (EHI): Information leakage (id. INL)
  Error handling (id. EHD)
- Secure code design (SCD): Variable types/operations (id. VTY)
  Declarations and initialisation (id. DIN)

5.2. Annex 2: Final Report Structure

The results of the code review audit will be presented on a report that will be distributed to all parties specified on the scope of the project. There are two reports, a full (detailed technical) report and an executive report.

5.2.1. Detailed Report

The full code review report will include detailed sections of all the findings identified, and recommendations to mitigate or solve them:

Final code review structure for the full technical detail report:

1. **Introduction**
   - *1.1. Context*
   - *1.2. Objective*
   - *1.3. Scope*
   - *1.4. Definitions and acronyms*
2. **Executive summary** (high-level summary)
   - *2.1. Analysis*
   - *2.2. Findings (checklist)*
3. **Methodology**
   - *3.1. Planning*
   - *3.2. Execution*
   - *3.3. Assessment*
   - *3.4. Reporting*
4. **Audit details**
   - *4.1. Initial considerations*
   - *4.2. Planning*
   - *4.3. Detailed tests*
5. **Recommendations**
   - *5.1. Detailed recommendation*
   - *5.2. Prioritisation*
6. **Conclusions**
7. **Annexes**

Overall, the structure must allow the reader to understand the motive, objectives, method followed and the results obtained, giving a full overview of the code review status.

5.2.2. Executive Report

The executive report will be a reduced version of the full report, omitting the detailed technical reports, and containing a reduced methodology description.

This report will be tailored considering one main premise: a high-level overview allowing the stakeholders to easily see the current security status of the code reviewed, the main vulnerabilities and weaknesses, as well as the recommended steps to increase the security of the solution.

5.2.3. Communication Results Formatting

The communication of the results of the code review will make use of the tools that have been selected for this methodology. These results will be formatted based on the CVRF v1.1 structure defined by MITRE.

This structure must contain at least the following main elements:

Table 23: Basic CVRF elements

Common elements	**Product tree (*details*)**
> Document title > Document type > Document published > Document tracking > Document notes > Document distribution > Aggregate Severity > Document references > Acknowledgements	> Full product name > Relationship > Product groups
	Vulnerability
	> Title, ID, Notes > Discovery & Release date > CVE / CWE / CVSS scores > Product status > Threats > Remediation's > References

5.3. Annex 3: Code Review Procedure

5.3.1. Planning Phase

Project Inputs: The inputs will be provided before the start of the project, using a wiki system, file hosting system, or similar options.

The project inputs are listed as follows:

- Client expectations (Scope, Time, Budget). The client the organisation which is going to provide the budget for the code review.
- Documentation: architecture diagram, use cases, functional designs, data flow diagrams, and any other documentation that can provide better understanding of the code structure and/or functionality to the code review team
- Code: Uncompiled source code. Optionally, assembled code parts could be provided to allow the analysis by some tools (e.g. FindBugs).
- Contact information: email, phone, availability and backup of all the interested/relevant stakeholders (sponsor/client and/or software owner.
- Previous security tests: code review, penetration testing or similar analysis.
- The following folder structure will be used to store all the information:
    - /1. Management/
        - /1.1 Contact Information/ (Client information, code owner information)
        - /1.2 Planning/ (Scope, Time, Efforts)
    - /2. Input/
        - /2.1 Documentation/
        - /2.2 Code/
        - /2.3 Previous security test reports/
    - /3. Execution/
        - /3.1 Working files'/
            - /ModuleX/ (It contains a register of the code files that composed the module, as well as if they are part of a batch. The code files or batches will have an **ID: ModX-FileY** or **ModX-BatchZ** and will be used as name of the working folders)
                - 'ModX-FileY'
                - 'ModX-FileV-Divided' (when a code file is quite long and it has more than 500 code lines, the file is divided into smaller files of 500 code lines or less, as explained in the 'Execution Phase' subsection)
                - 'ModX-BatchZ ' (a set of code files that have to be analysed together, as explained in the 'Execution Phase' subsection.)

- /InterModuleX/ (For those batches that are composed of different code files from different modules, it will be considered as a 'virtual' module. This mode is composed by the batch that contains all the code files that are part of the inter module batch).

  - /3.2 Working Report/

- /4. Output/

  - /4.2 Evidences and checks/

  - /4.1 Final Report/

Planning: Review the project, and estimate the efforts required based on the input received. This estimation will depend on several key factors, as follows:

- The selected code review method ('Managed mode', 'Defined mode', or 'Optimised mode').
- The documentation of the code (quality).
- The number of controls and categories to use.
- The complexity of the code review (low, medium, high), as explained below.
- The difficulty of the controls to analyse, as some controls are quicker to review than others.
- Time or deadline restrictions imposed by the client.

The complexity of the code review is estimated according to the number of categories of the controls that are applicable to the module (software) under test. The classification criteria is:

- If there are 1 to 4 applicable categories, then the complexity of the module analysis is "*low"*.
- If there are 5 to 7 applicable categories, the complexity of the module analysis is *medium*.
- If there are more than 7 applicable categories, the complexity of the module analysis is *high*.

In the event that a big application has to be analysed, an efficient option is limiting the scope by reducing the number of modules to analyse, especially regarding the manual review ('Defined mode' and 'Optimised mode'). If reducing the scope is required then, in our experience, from 5% to 15% of the code represents a trustworthy sample to manually review the more critical aspects of the application.

However, these estimations assume that the quality of the documentation is high. In the event that the document quality is different, the estimation will vary greatly:

- For medium-quality documentation, the time/effort requirements will increase around ~50%.
- For low-quality, or non-existent documentation, the time/effort estimations will have to be increased at least by 100%.

Taking into account the previous information, several management aspects are defined:

- The code review planning
- Software modules are going to be examined, which categories are included, what level of depth has been requested, etc.).
- Time

- Final budget of the project.

After the definition of the management aspects of the code review, the client (the organisation which is going to provide the budget for the code review) must approve those aspects. Thus, to sum up this phase there are several steps:

- First, collect the project input provided by the client.
- Analyse the project input obtained.
- Define the management aspects of the code review project.
- Get the approval of the client regarding the management aspects of the code review (final budget required, time and effort required).

5.3.2. Execution Phase

The project will start after the client approval has been obtained; nevertheless, several actions should be carried out beforehand:

- Organise the code review team staff and define their roles during the project.
- Prepare the roadmap based on the project planning and scope.
- Decide how the code files are going to be analysed using the code review controls. (Code Review Process)

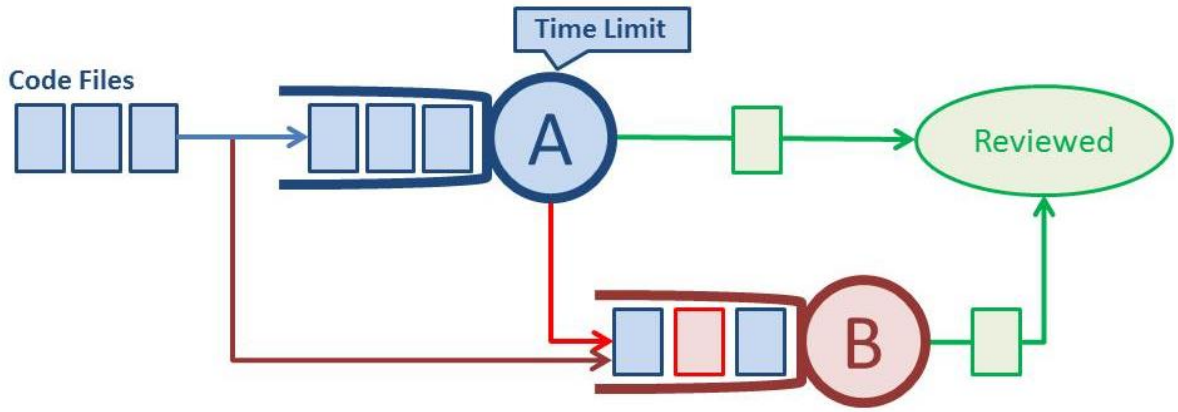
Team Organisation: Organising the team is an important matter that will directly affect the efficiency of the review process. To follow an efficient approach, our proposal is dividing the team into two sub-teams. The idea is to have two different queues of code files to analyse (‘A’ and ‘B’), where one of them will face those controls that are more difficult to analyse.

In order to do that, queue ‘A’ team will have a time limit per code file. If the analysis of the code file is not over before the time limit or its analysis is quite difficult, the current code file will be sent to queue ‘B’ team. Queue ‘A’ will go on analysing code files while queue ‘B’ will analyse the slow and/or difficult controls.

Most controls will be analysed faster, as they are not applicable in all code files, or are easy to detect, whereas the difficult ones are usually a minority. While not applicable ones are discarded and easy ones are analysed in queue ‘A’, queue ‘B’ is analysing the difficult ones.

Nevertheless, queue ‘B’ will be working similar to queue ‘A’ until a controls exceeds a time limit and is sent to queue ‘B’ for analysis. The time limit has to be determined, depending on the code review context, but a good approach will be 30 minutes per code file. The advantage of this method is that difficult controls are analysed in parallel, therefore increasing the efficiency of the code review process.

Figure 11: Working Queues



Once the team organisation is understood, the team has to be divided into those groups. Taking into consideration the nature of the queues, team 'A' can be composed by less experienced code reviewers while queue 'B' will be composed by experts and veteran team members.

This approach is not a static one, this model can be extended by adding more queues with different time limits, for instance:

- Queue 'A', time limit: 30 min per code file. If a code file exceeds the time limit, it will be sent to queue 'B'.
- Queue 'B', time limit: 1 hour per code file. If a code file exceeds the time limit, it will be sent to queue 'C'.
- Queue 'C', for those code files that are difficult to analyse, or they require more time than the time limits.

Nevertheless, it is important to notice that some security functions involve several code files instead of only one. In this case, all related code files will be composed a group (batch) of code files, and analysed together (in the same working queue, or same person depending on its particular case).

If a code file is too long (more than 500 code lines), the file will be divided into a set of files with a maximum length of 500 code lines approximately. These 'virtual' files will composed a batch of code files saved in same folder than the original one, and they will be analysed as a batch.

Code Review Roadmap: The roadmap is an exercise where the different tasks and actions are ordered during the code review, as follows:

- Launch the code review tool for the modules that will be analysed using the 'Managed Mode' of the methodology, and analyse the results. Anyway, the software modules under analysis using the 'Defined Mode' and 'Optimised Mode' will be analysed using the tool too, because the code review modes are incremental ones, the upper levels extend the lower levels.
- Those software modules, which are going to be analysed using the 'Defined Mode', have to be divided into the working queues, taking into account that queue 'B' has to deal with those code files that require larger time frames.
- In the event of analysing software modules using 'Optimised Mode', this will be carried out after analysing the modules under test using the 'Defined Mode'. That is why the 'Optimised Mode' is the 'Defined Mode' + specific controls. It would probably require more time to analyse the code

files under this mode, because of the larger number of controls. By this way, the rest of the code under analysis will be reviewed faster.

Taking into account the previous information, our proposal for the roadmap structure would be as follows:

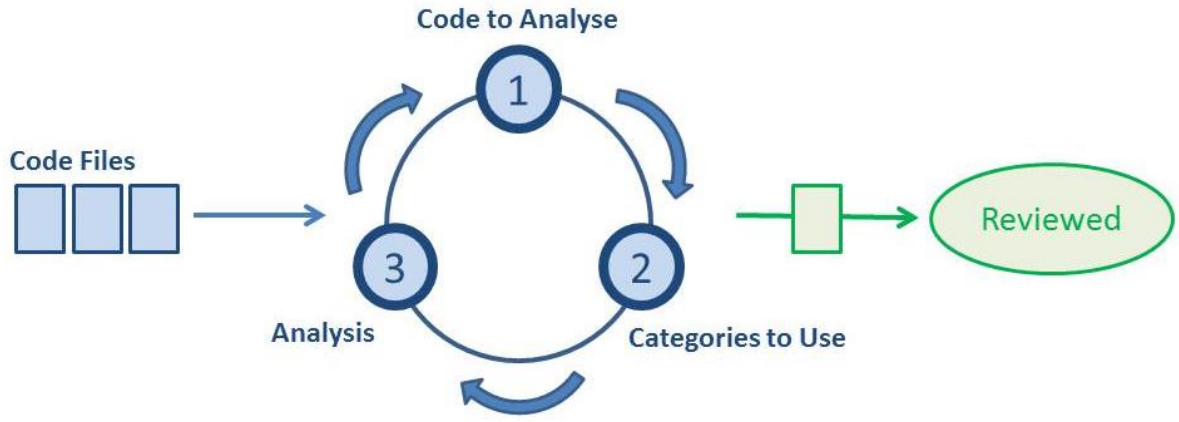
- Launch the tool to test all modules under scope.
- Review the results using the team structure defined previously (working queues).
- Analyse the modules under test using 'Defined Mode' (those that have to be analysed using the 'Defined Mode' as well as the ones of the 'Optimised Mode'), using the working queues. While no issues arise, the modules will be distributed indistinctly. If an issue arises, all software modules will be sent to queue 'B' until all the issues are solved.
- Once the 'Defined Mode' is done for the modules under test, the specific code review controls of the 'Optimised Mode' will be addressed (using the working queues).

Code Review Process: In order to be efficient during the code review execution, the code files have to be analysed in a structured way. Our proposal is an iterative approach where those parts of the code that are not clear to analyse will be left for the next iteration. By this way, the easy findings (about 80% of the code) will be done fast, leaving the blocking ones (about 20%) for the next iteration.

1. The first step to analyse the code file is to examine the file, highlighting what parts of the code are susceptible to be reviewed from a security point of view. Those parts that are not easy to determine if they have to be analysed, will be marked as "pending" and will be reviewed in the next iteration.
2. Once those parts are detected, the applicable categories are selected for each part. Those parts that are not easy to determine what categories have to be applied, will be marked as 'to be reviewed' and will be examined in the next iteration.
3. Analyse each part using the selected categories. Those parts that are not easy to analyse will be marked as "pending" and will be reviewed in the next iteration.

If all parts of the code have been analysed, a final iteration is required to verify that all parts have been analysed.

Figure 12: Code Review Process



To do that, a good idea is to use a support document to aid during the code review process. This document has to contain what part of the code is analysed (code lines and file name, e.g. '34:65), and what categories and sub categories are utilised.

Figure 13: Structure of the Tables of the Support Document

Code File	Code to Analyse	Categories	Current Status	Check in the next Iteration	Notes
FileX.z	34:65	xxxxxx yyyyyy	0/1/2/3/ Done	Yes/No	Abksdfkalsdfasdf

In the event of having a batch of code files, the previous table will have several entries for each code file. If many parts of the same code file have to be considered, several entries will be added for each part of the code to examine.

This support document will be quite important if a code file is sent from one queue to another, because this will reflect the last status of the analysis, avoiding wasting time doing the same things twice. It is desirable to have a modification history in those documents.

Thus, two documents have to be used during the code review process: the support document to track the analysis, and the excel file with the results of the analysis. Our proposal is generate a folder, in the project working directory, for each code file or batch of code files. In that folder, the two documents previously mentioned will be stored. The names of those files will be as follows (depending on whether is a code file or a batch under test.):

- '**Results_FileX.xml**' or '**Results_BatchX.xml**'.
- '**SupDoc_FileX.doc'** or '**SupDoc_BatchX.doc'**

Previously, in the '**Code Review Roadmap**' part we explained that the methodology modes will be analysed incrementally. The previous files will be reused for the different modes, but the support document will have a table for each analysed mode, and the modification history correctly updated.

Support document example:

support_document.docx

Once all the software is analysed, all results (excel files) have to be aggregated in a single excel file. This will be created in the folder 'Working Report', where all the results will be added and be ready for the assessment phase. This task will be done by a single code reviewer, to ensure the integrity of the results, and this person will have a checklist with all the code files and/or batches analysed in the code review. This person is in charge of adding the results will use that check list to ensure that all results are added.

5.3.3. Assessment Phase

Once the Execution Phase is finished and the results are put together in the Excel, we assess and evaluate the findings.

To carry out this task, the excel file provides a set of cells with a dropdown list with the values and their meanings contained in **Table 15** to automatically calculate the final values for Threat, Vulnerability and Impact of each check. These values will be used to automatically calculate the value of these characteristics for the Controls that contains the checks evaluated and, so, the Global value of the issue.

This can be done by only one person or the effort can be divided among different members of the team, but our recommendation is to assign the task to only one person, to maintain the coherence and criteria of the assessment.

The process must be conducted as follows:

- For each check that does not comply with the condition, there will be a set of cells with dropdown lists containing the possible values corresponding to the characteristics of the Threat, Vulnerability and Impact related to the occurrence of the issue (See Table 15 in this document).
- This will automatically calculate the values for Threat, Vulnerability and Impact on sheet “Controls” and so the Global Risk Value for the control that contains the checks evaluated.

If it´s necessary to divide the effort among different code reviewers, we propose the use of temporary files as the way as the working files in the Code Review Process of the Execution Phase. The names of these files will be as follows:

- ‘**Assessment_CategoryX.xls**’.

Then all the partial results must be put together

In the special situation of finding a critical vulnerability, the procedure defined with the stakeholder in the preparation of the planning phase of the methodology (Section 3.1.1) will be followed.

Then a prioritisation plan must be developed in order to define what issues must be solve first according to their severity. This prioritisation plan will help developers to know what the more critical issues that must be solved are. (See Figure 8 in this document for a Sample of Priority Levels).

5.3.4. Reporting Phase

Once the assessment phase is completed, the assessment results, findings prioritisation action plan and recommendations must be included in the Executive Summary Report and the Detailed Technical Report, which are part of the Final Report.

For this purpose, a template has been developed to generate instances of the Final Report, to include all the information about the code review, stakeholder, scope, etc., and its results:

Code Review Report

This template will be modified once all the controls have been finalised

The ‘Overview’ sheet of the excel file generated in the assessment phase will help with the Executive report, finding indicators placed in the Executive summary (See Figure 7 in this document).

To carry out this task, the excel file generated in the assessment phase can be included as an object or with the Final Report.

In addition, a detailed list of recommendations must be included in the Detailed Technical Report.

All the documentation generated must be disseminated to the distribution list defined in the planning phase, using the communication methods defined in that phase.