RESILIENCE

Grant Agreement 101079792, RESILIENCE PPP

Software Development Plan Template (SDPT)

Title of Deliverable:	Software Development Plan Template (SDPT)		
Deliverable Number:	D2.8		
Type of Data:	Report		
Lead Beneficiary:	KU Leuven		
Publishing Status	Public		
Last Revision Date:	15/11/2023	**by:**	Rudy Demo, FSCIRE
Verification Date:	21/11/2023	**by:**	Board of Directors
Approval Date:	[DD/MM/YYYY]	**by:**	[Name]
Document Name:	RESILIENCE_WP2_D2.8_SW_Dev_Plan_Template_01.00_FINAL		

Funded by the European Union

Change History

Version Number	Date	Status	Name	Summary of Main Changes
00.01	14/10/2023	DRAFT	Initial Draft	
00.02	14/11/2023	2nd DRAFT	Revised Text	Revised version to answer comments from KU Leuven Team members
00.03	15/11/2023	3rd DRAFT	Revised Text	Revision changes confirmed
01.00	20/11/2023	FINAL	Final version	Minor comments from BoD

Author(s)

Name	Beneficiary	Role
Rudy Demo	FSCIRE	WP2 Team member

Distribution List

Name	Beneficiary	Role
Public	All	

1 Table of contents

Acronyms

Acronym	Full Form
CI/CD	Continuous Integration/Continuous Delivery
DoD	Definition of Done
DVCS	Distributed Version Control System
GDPR	General Data Protection Regulation
IT	Information Technology
OWASP	Open Web Application Security Project
PEP	Python Enhancement Proposal
PP	Preparation Phase
PSR	PHP Standard Recommendation
RI	Research Infrastructure
R-Suite	RESILIENCE Service Suite
SCRUM	Not an acronym, but a framework for Agile
SDP	Software Development Plan
SDPT	Software Development Plan Template
SW	Software
UI	User Interface
UX	User Experience
WCAG	Web Content Accessibility Guidelines

Definitions

Term	Definition
In-kind contributions	Many RIs may rely on some forms of in-kind support. This can be related to the use of donated scientific equipment or the exploitation of time machine or personnel costs (this is particularly true for pan-European RIs, which rely heavily on in-kind contributions from national members). They can also refer to technical components or equipment supplied by one of the partners (e.g. one Member State as a share of its contribution) and made available in-kind to the RI. Source: StR-ESFRI Study. GUIDELINES ON COST ESTIMATION OF RESEARCH INFRASTRUCTURES

In-house contributions	In-house contributions refer to resources that are directly invested by the organization overseeing the Research Infrastructure. These contributions are typically financial but can also include resources like staff time, equipment, and facilities that are owned and operated by the RI itself. In-house contributions represent the internal commitment and investment of the RI into its own operational and project activities, as opposed to resources sourced from external partners or stakeholders.
R-Suite	R-Suite refers to a comprehensive collection of software tools, services, and applications collected, developed, and maintained under the RESILIENCE Research Infrastructure. This suite is designed to cater to the diverse needs of researchers in the field of religious studies. It encompasses various functionalities, ranging from data analysis and storage solutions to collaboration and communication platforms, all integrated to provide a seamless user experience.

List of Figures

No table of figures entries found.

List of Tables

No table of figures entries found.

1 Introduction

1.1 Objectives of this document

1.1.1 From the proposal

Extract from [R1]: "The Software Development Plan Template is elaborated to define all best practices for the development, testing and installation of software to be created and maintained within the context of RESILIENCE. The SDPT is foundational for the IT aspect of RESILIENCE, since each software project will then create an instance of this template specifically adapted to the software maintained."

1.1.2 Long-term objectives

The Software Development Plan (SDP) for RESILIENCE is an essential instrument that serves to delineate a solid, scalable, and secure structure for the provision of software products and services to the community of researchers in religious studies. Recognizing the heterogeneous landscape of research on religion, and RESILIENCE's objective to offer its community a high quality user experience, it is imperative that the tools and services developed under this umbrella be crafted with precision, adaptability, and reliability.

One of the prime considerations for the RESILIENCE SDP is to **ensure consistency and standardization across all (non in-kind) software projects for the RESILIENCE central infrastructure**. For in-kind projects, this document is simply to be considered as a recommendation. This approach ensures **a harmonious set of practices, methodologies, and conventions** are used throughout, paving the way for excellent collaboration and integration amongst various IT projects. This also ensures that as the diverse requirements of research emerge, RESILIENCE software development methodologies remain robust, yet flexible enough to cater to these evolving needs.

Another significant focus is the **security and privacy embedded within the software products**. The nature of religious studies can be sensitive, and RESILIENCE is committed to integrating, when necessary, advanced security protocols that prioritize the protection of both data and user privacy. As the RESILIENCE infrastructure expands and accommodates a broader audience of researchers and their projects, it is vital that **the software solutions designed are efficient in scaling**, thereby adeptly handling the growing user base and the associated surge in data.

Central to the RESILIENCE approach is the user. Every piece of software, **every tool, and every service is designed keeping the end-user in mind**. This user-centric approach translates to intuitive interfaces, easy-to-navigate documentation, and prompt support, aiming to streamline the research process fundamentally. Furthermore, this plan emphasizes the importance of **fostering a collaborative environment**. By building software that not only supports internal collaboration but also offers the capability to seamlessly integrate with external systems and databases, RESILIENCE becomes a hub for collaborative research in religious studies.

Continuous improvement is ingrained in the fabric of the SDP. By assimilating feedback from the research community and proactively anticipating future requirements, there's an ongoing commitment to refine and enhance the software offerings. This spirit of continuous improvement extends to **rigorous testing and quality assurance**. It is of utmost importance that the software released under this initiative undergoes

exhaustive testing, spanning functional, security, usability, and performance parameters, to **ensure its robustness and efficiency**.

Moreover, it's not just about developing software but ensuring its relevance and efficiency over time. Hence, **the plan deeply emphasizes the long-term sustainability of the software**. Comprehensive strategies for regular software updates, maintenance patches, and, if needed, complete overhauls are discussed, ensuring the software remains pertinent over time.

In conclusion, this SDP Template underscores RESILIENCE's willingness to deliver premier software solutions and services for the religious research community.

1.2 Discriminative Application of the Software Development Plan

While RESILIENCE will evaluate the in-kind contribution by doing basic checks on availability of helpdesk, user documentation etc., the Research Infrastructure (RI) will not have the resources nor the authority to perform a full evaluation - e.g. digging in to the code base and hosting infrastructure - of the offered in-kind. The SDP must be therefore discriminatively applied to effectively manage the diverse nature of resources within our RI. The application of the SDP takes into consideration the distinction between in-kind contributions and in-house created resources.

This section delineates the tailored application of the SDP to these factors.

1.2.1 Application of the SDP to in-kind contributions

- Contributors of in-kind resources are expected to consider the SDP as a recommendation by the RI. When submitting their contribution meta-data , they will be asked to which aspects of the SDP their contribution adheres to.
- The RI will not directly manage or store highly sensitive data from in-kind contributions. Contributors must ensure that data is properly anonymized before integration into the RI's systems.
- A disclaimer will be provided to contributors of in-kind resources, outlining the limited liability of the RI concerning the security of these resources and the obligation of contributors to meet a minimum recommended set of security standards. These standards include essential security measures such as virus-checked data before submission, document the level of respect of OWASP Application Security Verification Standard (level 1 to 3), usage of data encryption for data depending on their sensitivity, access control, and user authentication compatible with the RI recommendations, etc.

1.2.2 Application of the SDP to in-house created resources

- In-house created resources must fully adhere to the SDP and the security framework developed by the RI. This framework will include advanced security measures and controls tailored to the specific risks associated with these resources. Exception to this compliance might be defined by the RI Security Officer.
- Regular security audits and assessments will be conducted to ensure that in-house resources maintain the highest level of security integrity.
- In-house resources should not include sensitive data as the RI does not recommend hosting this data on the RI platform. For the storage of this data, researchers should consider using another platform respecting the regional and international regulations.

2 Software Development Plan Template

2.1 Purpose of the Document

[This subsection must outline the primary reason or objective behind creating the SDP. The author should explain why the document exists, its role within the broader RI, and how it helps guide the software development process. This is essentially a brief summary of the SDP's raison d'être.]

2.2 Scope

[The author will define the boundaries of the SDP. It will describe what the document will cover in terms of software functionalities, features, connection with the RESILIENCE RI software/services management plan and the parts of the project it addresses. Conversely, it might also be helpful to mention what the SDP does not cover, ensuring clarity on its limitations.]

2.3 Intended Audience

[The author provides a detailed profile of the primary recipients or users of the document including the interfaces of the RESILIENCE RI involved into the RI software/services management. This section ensures that the content is targeted, relevant, and appropriate for those who will most frequently refer to the SDP.]

2.4 Document Overview

[The author must provide a brief synopsis of the SDP's structure and content. It helps orient the reader, providing a roadmap of what to expect as they delve deeper into the document. This section should briefly touch upon the major sections and highlight the flow of information, making it easier for readers to navigate to areas of interest or importance.]

3 IT Project Overview

[The 'Project Overview' offers a panoramic view of the project, giving readers an understanding of the project's origins and what are the needs covered within the context of R-Suite (the RESILIENCE Service Suite), goals, key players, governance structure (in relation to the wider RESILIENCE RI SW governance), resource distribution and high-level activities planning. This section sets the stage of the software development plan.]

3.1 Background and Context

[The author provides a historical narrative, explaining the origins of the software project. Elaborate on the existing challenges or gaps within the R-Suite context that spurred the initiation of this project. If there have been previous attempts or solutions, discuss their outcomes and how this current endeavour differs or improves upon them.]

3.2 Project Objectives

[The author defines the objectives of the project. Clarify what the author aims to achieve within the R-Suite framework. Lay out specific, measurable outcomes, ensuring that the author can later assess the success of the project against these outlined objectives.]

3.3 Key Stakeholders

[In this section, the author should identify and describe all the significant parties or groups with an interest in the project. Detail who they are, what their role is, and why they are vital for the project. Highlight their influence and interest, ensuring a clear understanding of their stakes.]

3.4 Project governance and RESILIENCE governance interfaces

[The author will need to explain the project's governance structure, highlighting decision-making processes, roles, and responsibilities. Emphasize how this structure interfaces with the broader RESILIENCE RI Software governance, ensuring readers understand the synergy between the project and the overarching organization. Maybe a RACI table would be suitable within this section]

3.5 Resource Allocation

[The "Resource Allocation" section is pivotal for the smooth progression and success of the project. It serves as a detailed blueprint of how various resources are distributed and utilized, a topic of keen interest to key stakeholders. Project funders will closely scrutinize this section, seeking clarity on the financial investments, their distribution across project stages, and provisions for any contingencies. It's not just about the monetary allocation; the precise dedication of software tools, development modules, teams, and technologies, which directly ties into the R-Suite architect's domain, plays a fundamental role in achieving the project's objectives.

The infrastructure system administrator, on the other hand, will be keenly focused on the specifics regarding hardware, networking, and system resources. A clear breakdown of infrastructure requirements and potential third-party integrations is crucial to staff the right people at RESILIENCE side. Given the interconnected nature of projects within the RESILIENCE initiative, a transparent outline of shared resources is paramount to avoid overlap, reduce conflicts, and ensure that all projects are adequately resourced.]

4 Development Methodology

[The Development Methodology section outlines the structured approach guiding the project's progression. The author must delve into the chosen method, whether Agile, Waterfall, or hybrid, and explains its eventual alignment with the broader development of the RI goals and stakeholder expectations. The section underscores the iterative nature of development, emphasizing regular feedback, team synchronization, and adaptability, ensuring clarity on the software's developmental journey. In case the development occurs outside the RI, it is important to highlight here the synchronisation with the RI SW governance team(s)]

4.1 Development Approach (e.g., Agile, Waterfall)

[The development methodology adopted must be described here. In alignment with contemporary development paradigms and to foster a flexible, adaptive, and responsive software development environment, RESILIENCE RI primarily recommends the adoption of the Agile methodology, specifically leveraging the Scrum framework. Agile's principles resonate with the objectives of RESILIENCE RI: they advocate for collaboration, adaptability, and delivering value in incremental stages. While the traditional Waterfall model serves its purpose in more rigid, predefined projects, Agile's iterative nature aligns more closely with the unpredictable and evolving landscape of religious studies. Moreover, in instances where software development occurs external to the RI framework, synchronization with the RI SW governance teams is maintained, ensuring alignment with overarching objectives and stakeholder expectations.]

4.2 Phases of Development

[The author should describe here the software development phases. As example, under RESILIENCE RI, following the Agile and Scrum practices, a project would be dissected into distinct phases: Product Backlog Creation, Sprint Planning, Daily Stand-ups, Sprint Review, Sprint Retrospective. Describe also when the author would consider/request the feedback of RESILIENCE RI IT stakeholders.]

4.3 Iterative Development and Feedback Loop

[For RESILIENCE RI, the software evolves through repeated cycles of development, testing, and feedback. Each iteration, or sprint, results in a potentially shippable product increment, ensuring that the software remains aligned with user needs and the ever-evolving landscape of religious studies and RESILIENCE RI. One of Agile's cornerstones, the feedback loop, ensures that stakeholder feedback is integrated at regular intervals. This not only enhances the software's quality but also its relevance. The author should describe here the intertwining iterative development with consistent feedback, to ensure that its software products for RESILIENCE RI are dynamic, adaptive, and continuously refined to serve the Religious Studies community's needs best.]

5 Standards and Conventions

[The "Standards and Conventions" section in the Software Development Plan delineates the guidelines for coding and documentation to ensure consistency, readability, and maintainability. It emphasizes the importance of versioning and a structured review process, facilitating smooth collaboration among developers and enhancing user experience. This section is pre-filled with the RESILIENCE preferred standards and conventions. The author should feel free to add any standard and conventions adopted for its contributions, which should include a motivation of choice.]

5.1 Coding Standards

[Describe here what coding standards are adopted by the project and eventually add a link to the documentation describing their implementation. As examples of coding standards recommended by RESILIENCE RI we can mention:

- *PEP 8 for Python: This is the style guide for Python code. The Python community and many organizations that use Python adhere to these standards to ensure consistent and readable code.*
- *Google Style Guides: Google has its own set of style guides for various programming languages like C++, Java, Python, JavaScript, and others.*
- *Java: Oracle's Code Conventions: These are traditional guidelines for writing readable Java code.*
- *PSR Standards for PHP: The PHP-FIG organization has created several PHP Standard Recommendations (PSRs) to standardize coding practices for PHP.*
- *Microsoft's C# Coding Conventions: Microsoft provides guidelines for writing consistent, readable C# code.*
- *OWASP Secure Coding Practices: Apart from style, security is paramount. OWASP provides a checklist of secure coding practices for various languages and platforms.]*

5.2 Version Control

[Within RESILIENCE, we consider that Version control, within the framework of CI/CD, isn't merely about tracking code changes; it's the backbone that ensures seamless automation, immediate feedback, and rapid deployments. The intertwining of CI/CD principles with version control practices results in an efficient and error-resistant software delivery process.

- ***Distributed Version Control System (DVCS)**: Adopting a DVCS, such as Git, remains essential. It supports multiple parallel development streams via branching, allowing developers to work on features, hotfixes, or experiments without disrupting the main codebase.*
- ***Feature Branching Strategy**: In a CI/CD setting, the approach to branching becomes even more crucial. Each feature or user story should have its branch, ensuring isolated development and testing. Once the code is deemed ready, it can be merged into the main branch, triggering the CI process.*

- ***Commit Regularity and Clarity**: Regular, smaller commits are favoured in a CI/CD environment. They are easier to test, reducing the chances of integration issues. Every commit should have a clear and descriptive message, detailing what changes have been made and why.*
- ***Automated Builds and Tests**: Integrating changes from feature branches into the main branch should automatically trigger a build process. This ensures that the new code integrates well with the existing codebase. Automated tests, which should be part of the CI pipeline, immediately catch any issues, allowing for rapid feedback and correction.*
- ***Pull/Merge Requests and Code Reviews**: Before code gets merged into main branches, a pull/merge request should be raised, prompting a code review. This step, while a staple of good version control, is pivotal in CI/CD to prevent potential issues in the deployment pipeline. Automated checks, often run in the CI environment, can be associated with these requests to ensure code quality and compliance.*
- ***Environment-Specific Branches**: CI/CD often involves multiple environments, like development, staging, and production. Having branches that correspond to these environments ensures that the right code version is deployed in the right environment, facilitating staged rollouts and blue-green deployments.*
- ***Rollbacks**: One advantage of a robust version control system in CI/CD is the ease of rollbacks. If a deployment encounters issues in a particular environment, the system can quickly revert to a previous stable state using the version history.*

Describe here the principles adopted by the project.]

5.3 Documentation Standards

[While Agile emphasizes working solutions over hefty documentation, RESILIENCE RI consider that the latter remains crucial for clarity, future modifications, and onboardings. RESILIENCE RI's approach to documentation within Agile and Scrum context involves:

- ***User Stories and Acceptance Criteria**: Clearly written user stories with specified acceptance criteria help developers understand the goal, aiding in feature development and testing.*
- ***Code Comments**: while the code should be self-explanatory, crucial segments or complex algorithms benefit from explanatory comments.*
- ***API Documentation**: If the project involves developing APIs, tools like Swagger can automate documentation, ensuring it stays updated with each sprint.*
- ***Sprint Retrospectives and Reviews**: At the end of each sprint, document lessons learned, challenges faced, and potential improvements. This not only aids in the continuous improvement of the development process but also serves as a reference for future projects.*

- ***Project (or RESILIENCE RI) Knowledge Base**: Maintain a regularly updated knowledge base where developers can share insights, challenges, resolutions, and best practices, fostering a collaborative learning environment.*

Describe here the documentation standards adopted by the project.]

6 Architecture and Design

[The "Architecture and Design" section of the Software Development Plan provides a holistic view of the software's blueprint, covering the interrelation of its components within the SW itself but within the broader RI SW and IT services. It elucidates the system's overall design, data flow, modularity, and interface interactions. This roadmap gives developers a structured and clear direction, ensuring the respect of the RI software's quality indicators and alignment with the RESILIENCE goals.]

6.1 System Overview

[In this subsection, the author should provide a high-level summary of the system in its entirety. Considering Agile's user-centered focus, start by documenting the primary user roles and stories the system caters to. Then, provide a brief on the system's purpose, its main features, and how it fits within the broader RESILIENCE RI. Given Scrum's iterative nature, highlight any version or iteration details to show the system's evolution over time.]

6.2 Component Architecture

[Document the distinct components/modules of the software. Start by using tools like Component Diagrams that visually depict these components and their interdependencies. For each module, provide a brief description, its purpose, and its interactions with other modules. In line with Agile principles, maintain a backlog of components that are yet to be developed or refined. This dynamic document should adapt and grow with every sprint, reflecting the iterative development process.]

6.3 Interface Design

[This section should detail the user interfaces (UI) and any other system-to-system interfaces. Use mock-ups, wireframes, or UI prototypes to provide visual representation. Alongside each visual, provide a brief on its functionality, purpose, and any user stories it addresses. As Agile emphasizes feedback, ensure there's a mechanism to gather feedback on these designs for continuous improvement. In a Scrum context, these designs can evolve over sprints, reflecting user feedback and changing requirements.]

6.4 Data Flow and Management

[The author must document how data is processed, stored, and moved throughout the system by utilizing Data Flow Diagrams to visually represent this. Clearly define the data sources, storage mechanisms (like databases or cloud storage), data transformations, and endpoints. Also, provide details on how data integrity, consistency, and security are maintained. Given the Agile approach, always be ready to revise this section as more insights are gained from iterative development and testing phases.]

7 User-Centric Design and Usability (ONLY for SW components with UI)

[The "User-Centric Design and Usability" aims at harmonizing the usability and visual identity of RESILIENCE SW and IT services, prioritizes the end-users, emphasizing the significance of crafting software that's both functional and user-friendly.]

7.1 User Experience (UX) Principles

[Given RESILIENCE RI's commitment to modern IT practices, the UX principles should be documented iteratively, with each sprint or iteration considering user feedback and analytics. The author should start by detailing the primary user personas that will interact with the software, ensuring these personas are representative of the religious studies domain. Discuss the core objectives these users aim to achieve using the software and then outline the user journeys. As Agile and Scrum emphasize adaptability, ensure that the author incorporates avenues for regular feedback – perhaps at the end of each sprint – to refine these principles. By the end of the project, the author should have a well-defined set of UX principles that have been validated and refined over multiple iterations, ensuring the software is tailored to the real needs of its users.]

7.2 User Interface (UI) Guidelines

[The RI will at some point in time establish a set of UI guidelines to try and maintain consistency in interface design across all software components with UI under its management. The UI guidelines will have laid down the RI basic visual identity standards, like color schemes, typography, and iconography, that align with the RESILIENCE branding. Each project, based on its specificities should consider adopting a component-driven UI design approach allowing to design and develop reusable UI components that can be adjusted and refined throughout the RESILIENCE RI to ensure velocity of development and cost effectiveness. This section should incorporate feedback loops, similar to the UX section, to allow for the continuous refinement of these components. Ensure that the guidelines provide clear directions on how these components can be used and combined, facilitating both consistency and flexibility in design.]

7.3 Accessibility Standards

[Prioritizing accessibility is paramount to ensuring inclusivity. In this section, emphasize the necessity to adhere to global accessibility standards, such as the Web Content Accessibility Guidelines (WCAG). Given RESILIENCE RI's inclination towards modern methodologies, adopt an Agile approach to accessibility. Start with a basic set of accessibility criteria and then, with each sprint, expand and refine these criteria based on user feedback and testing. Incorporate tools[1] that can automatically test for accessibility during the development process, ensuring

[1] There are various tools and software used for testing the accessibility of IT services, such as:

- Automated testing tools like Axe, Wave, or Google Lighthouse, which can quickly identify some of the common accessibility issues.

that any new features or changes remain compliant. Highlight the importance of considering diverse user needs, ensuring the software is usable by people with various disabilities, and ensuring that the accessibility features are tested and validated by users with those specific needs.]

- Screen readers like JAWS, NVDA, or VoiceOver, used by visually impaired users, which help in testing how accessible a web service is for non-sighted users.
- Color contrast analyzers to ensure that text and background combinations meet the required contrast ratios.

8 Security and Privacy

[The "Security and Privacy" section underscores the RESILIENCE commitment to robust security mechanisms and data integrity. Emphasizing advanced security technologies, ethical data handling, and strict regulatory compliance, the section integrates with and references the RESILIENCE Security Management Plan document, ensuring that software development is aligned with overarching security objectives. Through this harmonization, the project showcases its dedication to safeguarding user trust, data confidentiality, and the overall integrity of the software amidst the evolving landscape of digital threats.]

8.1 Security Protocols

[In this subsection, the author must describe the specific security methodologies and practices that the development teams adopt. Given RESILIENCE RI's inclination towards Agile and Scrum, highlight how security is ingrained in the iterative development process. Discuss tools like continuous integration and continuous delivery (CI/CD) pipelines that emphasize security checks and balances at every stage. Mention techniques such as "shift left" which promotes early-stage vulnerability detection in the Agile development cycle. Outline any automated tools or manual code reviews to detect security flaws and how findings are prioritized and tackled in the Scrum backlog.]

8.2 Data Protection and Encryption

[The author should detail how data is protected both at rest and in transit. Given the iterative nature of Agile, elucidate on how feedback from each sprint informs data protection strategies in subsequent sprints. Discuss the encryption methods employed – asymmetric, symmetric, or hashing – and how the choice of a particular method aligns with user stories or features being developed. Emphasize practices in RESILIENCE RI that ensure encrypted data remains confidential, and elaborate on how decryption keys are managed, stored, and rotated.]

8.3 Compliance and Regulations

[Even in the fast-paced world of Agile and Scrum, compliance can't be overlooked. In this section, the author delves deep into how each sprint or iteration incorporates tasks that ensure regulatory compliance. For instance, when GDPR or any other data protection regulation is concerned, explain how user stories or features are developed in line with these guidelines. Highlight any automated tools or processes that assist teams in staying compliant. Describe the "Definition of Done" in Scrum tasks that encompass compliance checks.]

8.4 User Privacy and Confidentiality

[Given the user-centric nature of Agile and Scrum, the author must describe how user privacy is at the forefront of RESILIENCE RI's development practices. Discuss practices like data anonymization, pseudonymization, and tokenization. Describe how user stories are crafted with privacy considerations in mind. For instance, if a feature involves collecting user data, explain how the team ensures only the minimal required data is collected, stored, and processed. Highlight feedback loops from privacy audits or user feedback that feed into the Scrum backlog, ensuring continuous improvement in safeguarding user privacy.]

9 Testing and Quality Assurance

[The "Testing and Quality Assurance" section in the RESILIENCE Software Development Plan should emphasize a rigorous processes and methodologies employed to ensure software reliability and excellence and the respect of RESILIENCE RI Software Quality Criteria. Covering the extensive gamut from individual unit tests to overarching quality standards and reviews, this section communicates the project's unwavering commitment to delivering software that consistently meets or exceeds the established quality benchmarks.]

9.1 Testing Strategies

[In this section, the author should detail the specific testing methodologies that will be used throughout the software development process. Given the Agile and Scrum orientation, continuous integration and continuous testing should be highlighted and speaking about the frequency of testing sprints, how they align with development sprints in Scrum, and the use of automated testing tools that facilitate rapid testing cycles. Also, touch upon how feedback from these tests will be integrated back into the development cycle, ensuring iterative improvements.]

9.2 Test Cases and Scenarios

[Here, the emphasis should be on creating a comprehensive suite of test cases that cover the entirety of the software's functionality. Given the Agile framework, the test cases should be dynamic and evolve with each sprint. The author should describe how user stories and requirements from the backlog are converted into specific test cases. Outline the process for ensuring that test scenarios cover positive, negative, edge cases, and any other crucial use-case scenarios. Detail how these scenarios will be updated, stored, and managed across different sprints in the Scrum cycle.]

9.3 Performance Testing

[Performance testing in an Agile environment often requires regular benchmarking to ensure that new features or changes don't degrade the software's performance. The author should describe how performance metrics will be established and how performance tests will be scheduled. Given the iterative nature of Agile, it's essential to address how performance regressions, if found, will be prioritized in the product backlog. Also, highlight the tools and platforms that will be used for load testing, stress testing, and scalability testing.]

9.4 Security Testing

[Security is paramount, especially in the realm of religious studies where sensitive data (ie https://soarproject.eu/ protecting places of worship in Europe) might be at play. The author should detail how security tests will be integrated into the Agile development cycle. Describe the regularity of security audits, vulnerability assessments, and penetration tests. Given Agile's rapid cycles, mention the utilization of automated security testing tools that can quickly identify vulnerabilities. Additionally, emphasize how security findings will be prioritized in the Scrum backlog and the process for rectifying identified security issues.]

10 Approval and Sign-off

[The "Approval and Sign-off" section in the RESILIENCE Software Development Plan marks a critical point of the Software LifeCycle ensuring that the software has satisfied all RESILIENCE Quality Indicators.]

11 Collaboration and Integration

[The "Collaboration and Integration" section underlines the importance of synergy among diverse teams (and/or the SW development team and the RESILIENCE SW governance team) and the seamless merging of various software components. This section must explain how the SW development team ensures an harmonious developmental environment and a smooth interfacing with external systems by highlighting tools, practices, and strategies for effective communication and (system) integration.]

11.1 Collaboration Tools

[With an inclination towards modern IT development approaches like Agile and Scrum, the documentation for this section should focus on tools that enhance team collaboration, boost productivity, and maintain the Agile ethos.

- ***Detail the tools**: Enumerate and briefly describe the specific collaboration tools used, whether for task tracking (like Jira or Trello), code sharing (such as GitHub or Bitbucket), communication (like Slack or Microsoft Teams), or documentation (Confluence, Google Docs).*
- ***Guidance on usage**: Provide a brief guideline on how each tool should be used within the project, laying out best practices to ensure effective communication, tracking, and reporting.]*

11.2 Integration Points and APIs

[Given that RESILIENCE RI involves multiple software components, this section should address how these elements interact.

- ***Detailing the integrations**: For every major component or system the software interacts with, describe the nature of that interaction. This might include data exchanges, functional dependencies, or user interactions.*
- ***APIs**: Discuss the APIs in use or being developed. Detail their purpose, the kind of data they handle, and their protocols. Given the Agile focus, also mention any API documentation tools or platforms, like Swagger or Postman, to ensure continuous updates and clarity for developers.]*

11.3 Third-Party Software and Licences

[This section should provide clarity on external software utilities integrated into the project and the legal implications tied to them.

- ***List the software**: Enumerate each third-party software, tool, or library used.*
- ***Purpose of use**: For each, provide a brief description of its role in the project. This could be a particular function it performs, a problem it solves, or a feature it adds.*
- ***Licensing**: For every third-party software mentioned, detail its licensing type. Highlight any obligations or limitations these licenses impose on the software's usage, redistribution, or modification. Given the Agile and iterative nature of development, also stress the importance of regularly checking licensing terms in case of updates, ensuring compliance at all stages.]*

12 Training and Documentation

[The "Training and Documentation" section must fully support the RI's commitment to fostering an empowered and informed researchers' community. By emphasizing comprehensive, user-friendly documentation and tailored training sessions, this section ensures that stakeholders can navigate, understand, and make the most of the software, leading to not only widespread adoption but also optimized utilization. See RESILIENCE RI Training Management Plan for more information about RESILIENCE training and documentation Quality Indicators]

12.1 End-User Documentation

[Given RESILIENCE RI's focus on religious studies, end-user documentation should be user-centric and simplified to cater to a diverse user base, some of whom might not be tech-savvy.

- ***Nature of Content**: Detail step-by-step instructions, frequently asked questions (FAQs), screenshots, and walkthroughs. Use case scenarios from a religious studies perspective would also help in contextualizing the software's usage.*
- ***Iterative Approach**: Given the Agile methodology, documentation should be iterative and updated as features evolve. Every sprint or iteration might introduce changes that users need to be aware of.*
- ***Feedback Loop**: Integrate a feedback mechanism where end-users can highlight areas where they faced difficulties, ensuring that future documentation becomes more targeted.*
- ***Format & Accessibility**: Offer both online (searchable web pages, downloadable PDFs) and offline modes (printed manuals), ensuring easy accessibility for all user types.*

Describe here the end-user documentation available and its characteristics.]

12.2 Developer Documentation

[This is meant for the internal team and any third-party (ie the RESILIENCE RI itself) developers who might collaborate or build upon the project in the future.

- ***Code Documentation**: Emphasize inline code comments, explaining the logic and purpose of code segments. This is especially crucial in Scrum where teams might rotate, and new developers need to get onboarded swiftly.*
- ***APIs and Integration Points**: If the software has any APIs or third-party integration points, provide detailed documentation about endpoints, methods, request-response structures, etc.*
- ***Agile Artifacts**: Document the product backlog, sprint logs, user stories, and acceptance criteria. This provides clarity about development decisions and the roadmap.*
- ***Development Environment Setup**: Guide developers on setting up their development environment, detailing required tools, version controls, and dependencies.*

Describe here the developer documentation available and its characteristics.]

12.3 Training Materials and Workshops

[Training is essential for both end-users/researchers and developers to ensure optimal software usage and development.

- ***Modular Training Content**: Given the sprint-wise development in Agile, training materials should be modular. As features get rolled out, relevant training modules can be introduced or updated.*
- ***Interactive Workshops**: Organize hands-on workshops after significant releases. Use Scrum's review meetings as a point to introduce these workshops, ensuring immediate and relevant training.*
- ***Online Resources**: Offer webinars, video tutorials, and interactive quizzes. This supports asynchronous learning, allowing users and developers to learn at their own pace.*
- ***Feedback Mechanism**: After every training session, gather feedback. This helps in refining training materials and understanding areas that need more emphasis in future sessions.*

Describe here the training material available and its characteristics.]

13 Maintenance and Sustainability

[The "Maintenance and Sustainability" section must detail strategies for regular updates, adaptability, and long-term relevance, it showcases the project's commitment to not only maintaining but also evolving the software to ensure its continuous alignment with RESILIENCE RI users' needs, technological shifts, and broader stakeholder expectations.]

13.1 Maintenance Strategy

[In the Maintenance Strategy section, the author should define the methodologies and practices that will be employed to address routine maintenance tasks and to fix bugs or issues that may arise post-deployment. Given RESILIENCE RI's preference for Agile and Scrum:

- *Discuss how maintenance will be incorporated into the Agile framework. For instance, will there be dedicated sprints for maintenance, or will maintenance tasks be integrated into regular sprints?*
- *Detail how reported issues will be triaged. With Agile, prioritize user feedback and rapidly integrate it into subsequent sprints.*
- *Mention tools or platforms to be used for bug tracking, issue reporting, and how these integrate with the author Agile management tools.]*

13.2 Update Cycles

[For the Update Cycles section, the author should:

- *Define the frequency and nature of software updates. Agile promotes iterative development and continuous integration, so specify if updates will follow a set cycle (like every two weeks at the end of a sprint) or if they will be more dynamic.*
- *Discuss how updates will be communicated to stakeholders, especially users/researchers who rely on the software. Transparency is key in Agile methodologies.*
- *Explain any downtime, if required, for updates, and how this will be minimized or scheduled to have the least impact on users/researchers.]*

13.3 Long-term Sustainability Plan

[In the Long-term Sustainability Plan, the author should:

- *Discuss how the software will adapt to future technological changes and shifts in the domain of religious studies. Given the iterative nature of Agile, emphasize the flexibility to incorporate new technologies, methodologies, or functionalities as they emerge.*
- *Mention any plans for scaling the software, be it to accommodate more users, integrate more data sources, or expand in terms of functionalities.*
- *Detail strategies for securing continuous funding or resources for the software's long-term development, including potential partnerships, grants, or other financial strategies.*

- *Highlight training or upskilling plans for the team to ensure they're always equipped with current knowledge, aligning with the principles of continuous learning in Agile.]*

14 Risks and Mitigations

[The "Risks and Mitigations" section of the Software Development Plan emphasizes the identification of potential challenges and the proactive strategies in place to counter them. Covering a spectrum of risks, from technical to external, it showcases the project team's foresight and adaptability, offering stakeholders assurance of the project's resilience and preparedness for any uncertainties.]

14.1 Identified Risks

[In this section, the author should aim to present a comprehensive list of potential risks that the project might encounter. Given RESILIENCE RI's favouring of Agile and Scrum methodologies, it's essential to adopt an iterative and dynamic approach to risk identification:

- ***Dynamic Risks Log**: Given Agile's iterative nature, maintain a dynamic risks log, updated after each sprint or iteration. Each risk should be clearly defined with a brief description.*
- ***Categories of Risks**: Organize risks into categories. For instance, Technical Risks (related to technologies, tools, or platforms used), Operational Risks (related to daily operations, like availability of team members), and External Risks (factors outside the project's control, such as regulatory changes in religious studies).*
- ***Severity and Probability**: For each risk, gauge its severity (impact if the risk materializes) and its probability (likelihood of occurrence). This helps in prioritizing risks.]*

14.2 Mitigation Plans

[The essence of this section lies in outlining plans that counteract the identified risks:

- ***Agile Risk Mitigation Strategies**: Utilize strategies aligned with Agile and Scrum. For example, use backlog prioritization to address high-severity risks early in the project lifecycle or allocate time in sprints for tackling potential technical debt that could lead to future risks.*
- ***Responsibility Assignment**: For each risk, assign a team member or a group who would be responsible for implementing the mitigation strategy. Given the collaborative nature of Agile and Scrum, encourage team discussions on devising these strategies.*
- ***Contingency Plans**: Some risks might be too uncertain to have a direct mitigation strategy. In such cases, create contingency plans. These are "Plan Bs" that are activated if a risk materializes.]*

14.3 Monitoring and Updating

[In line with Agile's emphasis on feedback and iterations:

- ***Review and Update**: At the end of each sprint or Agile iteration, conduct a risk review session. Assess the current risks, update their probability and severity based on recent developments, and introduce new risks that might have emerged.*

- ***Stakeholder Communication**: Maintain transparency with stakeholders about the project's risk landscape. Use tools like a risk burn down chart, which is in line with Scrum's visual communication tools, to depict how risks are being addressed over time.*
- ***Lessons Learned**: After addressing a risk or at the conclusion of the project, document lessons learned. This is not just a retrospective on what went wrong but also what strategies worked in mitigating risks. This knowledge base becomes invaluable for future projects within the RESILIENCE RI framework.]*

15 Conclusion, lessons learned and next steps

15.1 Conclusion

[Start this section with a succinct wrap-up of the Software Development Plan, highlighting the primary goals and objectives outlined within the document. Touch upon how methodologies were infused throughout the stages of planning and development, emphasizing adaptability, iterative progress, and stakeholder feedback.

15.2 Lessons Learned

[Here, the author should delve into a retrospective evaluation including:

- ***Sprints Retrospective**: Since Agile and Scrum revolve around iterative sprints, mention key takeaways from sprint retrospectives. Highlight what went well, what posed challenges, and how the team adapted or plans to adapt in the future.*
- ***Stakeholder Feedback**: Document key feedback or insights received from stakeholders during review meetings, emphasizing how they influenced or will influence future actions.*
- ***Technical Challenges**: Outline unforeseen major technical hurdles encountered during the development process and how they were addressed. This not only serves as a point of reflection but also as a guide for future projects.*
- ***Team Dynamics and Collaboration**: Reflect on the team dynamics, the efficacy of communication channels, and how collaborative tools and techniques fared in the Agile setup.]*

15.3 Next Steps

[This section should act as a forward-looking segment:

- ***Iterative Improvements**: Based on the lessons learned, lay out the planned iterative improvements for the next phases of the project. This showcases the Agile spirit of continuous enhancement.*
- ***Future Releases**: Highlight upcoming release plans, outlining any significant features or modules that are set for development.*
- ***Training and Upskilling**: If the lessons learned pointed towards any skill gaps or areas of improvement in the team, mention any planned training sessions or workshops to bridge those gaps.*
- ***Stakeholder Engagements**: Outline any upcoming stakeholder meetings, reviews, or feedback sessions that are scheduled, ensuring transparency and continuous alignment with stakeholder expectations.]*

16 Reference Documents

Reference documents are intended to provide background and supplementary information.

ID	Date	Title/Reference
R1	18/08/2022	GRANT AGREEMENT Project 101079792 — RESILIENCE PPP

Funded by the European Union

[end of document]