

FACULDADE DE ENGENHARIA DA UNIVERSIDADE DO PORTO

Improving Agility in Critical Software Systems Development

Joaquim Manuel Silva Cardoso Rodrigues



Mestrado em Engenharia Informática e Computação

Supervisor: Prof. Ademar Aguiar

Second Supervisor: J. Eduardo Ferreira Ribeiro

July 31, 2022

Improving Agility in Critical Software Systems Development

Joaquim Manuel Silva Cardoso Rodrigues

Mestrado em Engenharia Informática e Computação

Approved in oral examination by the committee:

President: Prof. Filipe Correia

External Examiner: Prof. António Rito Silva

Supervisor: Prof. Ademar Aguiar

July 31, 2022

Abstract

Agile Methods are a reaction to traditional ways of developing software and acknowledge the need for an alternative to documentation-driven, heavyweight software development processes [50]. Technology moves too fast, and as a result, there is the constant search for new and different techniques that can cover the values and principles of software development and are established with agile methodologies.

Critical software systems, such as those for aerospace, can be described as systems whose functional failures can lead to death, injury, occupational illness, damage to or loss of equipment/property, or environmental damage [11]. Therefore, the systems are usually highly costly since they need to ensure uninterrupted reliability, conformity to standards, and respective certification.

The currently applied methodologies are mainly plan-driven which within critical systems documentation management impose a challenge to the process as a whole. Documentation is essential for providing the safety quality, i. e., specifications, analyses, reviews, and testing carried out during the development of a system provide evidence of safety in the development of safety-critical software [43]. Additionally, it should follow specific standards and is vital in stakeholder communication, usually based on formalized processes, entailing the need for traceability and flexibility.

Despite the use of documentation being considered the primary challenge to Agile Methods application in critical software systems [17], its usage would be of particular interest to improve changeability while providing efficiency and effectiveness to all the phases of software development.

In this work, we aim to create mechanisms for automating document processing and management, improving the efficiency and effectiveness of the whole software development process of critical software systems, most concretely in the aerospace domain. The implemented tools are co-designed and validated iteratively in the concrete context of Critical Software [39] projects, within the spectrum of the PhD research from J. Eduardo Ferreira Ribeiro [33]. These produced tools are not qualified in compliance with DO-330 standard (Tools Qualification). Nevertheless, DO-330 standard is out of the scope of this research.

Given the aforementioned objective, we classify the procedure checklist of the DO-178C standard, Appendix A, pointing the types of automation possible in these checklist procedures. Given this classification, a Software Configuration Index Report Generation Tool and a Document Traceability Automation Tool were designed and implemented to fulfill our goal.

To validate our solution, an interview process was conducted with Critical Software's professionals, collecting feedback on the implemented tools and insights on future work. The tools were also target of synthetic tests, that allow us to conclude that document automation is possible in the critical-safety software development industry and carries several benefits with it.

Keywords: Agile, Critical Software, DocOps, Documentation, Certification

Resumo

As metodologias Agile constituem uma reação às formas tradicionais de desenvolver software e reconhecem a necessidade de alternativas aos complexos processos de desenvolvimento de software orientados à documentação [50]. A tecnologia evolui rapidamente e, por conseguinte, existe uma constante procura de novas e diferentes técnicas que cubram valores e princípios do desenvolvimento de software, estabelecidas com metodologias Agile.

Sistemas de software crítico, como os aeroespaciais, podem ser descritos como sistemas cujas falhas funcionais podem levar à morte, lesão, doença ocupacional, dano ou perda de equipamento/propriedade, ou dano ambiental [11]. Deste modo, estes sistemas são geralmente de alto custo dada a necessidade de garantir fiabilidade ininterrupta e conformidade com as normas e respetiva certificação.

As metodologias aplicadas atualmente são maioritariamente orientadas ao plano, o que constitui um desafio ao processo no contexto da gestão de documentação de sistemas críticos. A documentação é essencial para fornecer a qualidade de segurança, por exemplo especificações, análises, revisões e testes realizados durante o desenvolvimento de um sistema de software fornecem as provas de segurança no desenvolvimento de software crítico [43]. Adicionalmente, a documentação deve seguir normas específicas e é pivotal na comunicação entre stakeholders, geralmente baseada em processos formais, implicando a necessidade de rastreabilidade e flexibilidade.

Apesar do uso e gestão da documentação ser considerado um desafio primário à aplicação de metodologias Agile em sistemas de software crítico [17], a sua inclusão seria de interesse particular para melhorar a mutabilidade, simultaneamente com o incremento de eficiência e eficácia em todas as fases de desenvolvimento de software.

Neste trabalho, apontamos à criação de mecanismos para automatizar o processamento e gestão de documentos, melhorando a eficiência e eficácia de todo o processo de desenvolvimento de software de sistemas de software críticos, mais concretamente no domínio aeroespacial. As ferramentas implementadas são co-projetadas e validadas iterativamente no contexto de projetos da Critical Software [39], no espectro da investigação de PhD do J. Eduardo Ferreira Ribeiro [33]. Estas ferramentas não são qualificadas em conformidade com a norma DO-330 (Qualificação de Ferramentas). No entanto, a norma DO-330 está fora do escopo desta investigação.

Dado o objetivo supramencionado, procedemos à classificação da lista de processos da norma DO-178C, Anexo A, indicando os tipos de automação possíveis nestes procedimentos. Com esta classificação, uma Ferramenta de Geração do Software Configuration Index Report e uma Ferramenta de Automação da Rastreabilidade de Documentos foram projetadas e implementadas para alcançar o nosso objetivo.

Para validar esta solução, foi conduzido um processo de entrevistas com profissionais da Critical Software, recolhendo feedback acerca das ferramentas implementadas e perceções acerca do trabalho futuro. As ferramentas foram, também, alvo de testes sintéticos que permitiram concluir que a automação da documentação é possível na indústria do desenvolvimento de software crítico

e acarreta diversos benefícios com a sua adoção.

Palavras-chave: Agile, Software Crítico, DocOps, Documentação, Certificação

Acknowledgements

Firstly, I would like to express my gratitude to my supervisors, Professor Ademar Aguiar and J. Eduardo Ferreira Ribeiro, for providing valuable guidance and insights throughout this dissertation. Thanks should also go to Critical Software (and its professionals) for the help during this dissertation and the valuable feedback.

Secondly, words cannot express my gratitude to my sister, Francisca, and my parents, Maria and Floriano, for all the love and support throughout my life. A special thank you to my godparents, Maria and Francisco, and my late aunt, Laura, for teaching me the value of hard work and overcoming adversities.

Lastly, I would like to thank my friends and colleagues for all the moments of fun and laughter and for helping me become the person I am today.

Joaquim Rodrigues

“I am a man of fortune, and I must seek my fortune”

Henry Avery

Contents

1	Introduction	1
1.1	Context	1
1.2	Motivation	2
1.3	Problem	2
1.4	Objectives	3
1.5	Document Structure	3
2	Background	5
2.1	Agile	5
2.2	Critical Software Systems	7
2.3	DO-178C Standard	8
2.4	DocOps	9
3	State of the Art	10
3.1	Methodology	10
3.1.1	Literature Research Questions	10
3.1.2	Process	11
3.1.3	Databases	11
3.2	Agile in Critical Software Systems	11
3.2.1	Benefits (LRQ1)	12
3.2.2	Concerns (LRQ2)	13
3.3	Documentation Automation (LRQ3)	16
3.4	DO-178C Standard and Documentation (LRQ4)	18
3.5	Summary	18
4	Problem Statement	20
4.1	Current Issues	20
4.2	Scope	21
4.3	Requirements	21
4.4	Hypothesis & Research Questions	21
4.5	Methodology	22
5	Software Development Process Automation	23
5.1	Context	23
5.2	Approach	24
5.2.1	Manual Automation	24
5.2.2	Software Automation	25
5.2.3	Pipeline/Tests Automation	25

5.2.4	Scripting Automation	26
5.3	Automation Analysis	27
5.4	Summary	27
6	Solution Proposal	29
6.1	Introduction	29
6.2	Software Configuration Index Report Automation	30
6.2.1	Critical Software Process Overview	30
6.2.2	Implementation	31
6.3	Document Traceability Automation	35
6.4	Known Limitations	37
6.5	Implemented Work and the DMT	38
6.6	Summary	39
7	Evaluation	40
7.1	Experimental Setup	40
7.2	Results & Discussion	41
7.2.1	Correctness	41
7.2.2	Maintainability	41
7.2.3	Performance	42
7.3	Hypothesis and Research Questions Evaluation	43
7.4	Summary	45
8	Conclusion	46
8.1	Conclusions	46
8.2	Future Work	47
	References	49
A	DO-178C Standard Classified Checklist	53
B	SCIR Template	83
C	Generated SCIR	86
D	Document Traceability Tool - Log File	89
E	CSW Interview Guide	91

List of Figures

1.1	Expected DMT	3
2.1	Agile manifesto values	5
2.2	For how long Agile has been adopted, extracted from [13]	6
2.3	Types of industries using agile methods, extracted from [12]	6
2.4	Types of Critical Software Systems, extracted from [20]	7
2.5	DO-178C standard and its supplementary documents, extracted from [49]	8
2.6	DocOps life cycle	9
3.1	Architecture for model-based systems engineering documentation workflows, extracted from [3]	17
3.2	DO-178C standard required certification data, extracted from [29]	19
5.1	Waterfall Software Development Life Cycle, extracted from [7]	24
5.2	CI/CD Pipeline, extracted from [6]	26
5.3	Scripting Automation evolution into Pipeline/Tests Automation, adapted from [16]	26
5.4	Software Automation in this dissertation's context, adapted from [16]	27
6.1	The software development process under development within a PhD research [33]	30
6.2	SCIR Generator architecture	32
6.3	Document Traceability Automation Flowchart	36
6.4	Current state of the DMT	39
7.1	SCIR Generation Tool synthetic tests performance	43
7.2	Document Traceability Automation Tool synthetic tests performance	44

List of Tables

3.1	Key benefits of Agile practices adoption, adapted from [24]	13
3.2	Key concerns of Agile practices adoption, adapted from [24]	15
3.3	Concerns to Agile implementation in critical software systems software development	15
5.1	Characterization of the DO-178C standard checklist automation types	27
6.1	Gathered data from each endpoint after parsing process	33

Abbreviations

CSW	Critical Software
DMT	Document Management Tool
SCIR	Software Configuration Index Report
PBI	Product Backlog Items
V&V	Verification and Validation

Chapter 1

Introduction

This chapter describes the context and the problem at hand and the goals this work aims to accomplish. Section 1.1 presents the context of this work, while Section 1.2 explains the motivation. Section 1.3 describes the problem under study and its details and is followed by Section 1.4, detailing the expected goals of this work. Finally, Section 1.5 describes the document’s structure.

1.1 Context

Agile and its methods have continuously spread in different industries [13] since the Agile Manifesto [8] in 2001. Agile methods are adaptive, iterative and incremental, and people-oriented, providing an adaptive process to control software development’s unpredictability [46]. These methods are classified as a reaction to the traditional approaches to software development and its associated bureaucracies. Agile methods provide multiple benefits to the software development process [24, 44] (detailed and specified in Sections 2.1 and 3.2), including better flexibility, requirements changeability, and improved project efficiency.

Critical software systems are generally classified as highly costly since they need to ensure uninterrupted reliability, conformity to standards, and respective certification. Given Agile’s benefits, its usage is of particular interest in critical software systems development to improve changeability while providing proficiency and efficacy to all the phases of software development [46].

Critical Software (CSW)¹ develops and validates critical software systems for major organizations and uses agile software development methods for their projects, influencing their clients’ adoption [33]. Furthermore, it is of the company’s particular interest to broaden the Agile practices usage, given the results obtained so far.

¹<https://criticalsoftware.com/>

1.2 Motivation

The adoption of Agile practices has been growing since the Agile Manifesto was published, especially in the software development industry, due to the increasing efficiency and effectiveness of the team and delivered results.

While large software development companies such as Spotify² and Netflix³ were born Agile [34] and thrived in their respective areas, critical software development teams still incline towards plan-driven development strategies, such as Waterfall, considering them as standards [44].

Given the excellent result of Agile adoption in the IT and software development industries, there seems to be potential for its inclusion within the critical systems software development context. Bearing that in mind, this work intends to contribute for that purpose, concretely in the documentation management and, consequently, in the verification and validation (V&V), and certification phase of the critical software development.

This dissertation is also inserted in the line of investigation between Faculdade de Engenharia da Universidade do Porto and CSW, being conducted in collaboration with J. Eduardo Ferreira Ribeiro's PhD thesis [33].

1.3 Problem

Although Agile software development has gained enormous success in all kinds of product and system development, the development of critical software still needs a lot of work to implement Agile methods [45]. One of the obstacles to this adoption (as we elaborate in Section 4.1) is the certification phase [38], where the software is tested, and the produced documentation during the development process is validated and certified by responsible authorities.

Taking this into account, our problem can be divided into four different questions:

- How can agile practices help to improve the efficiency and effectiveness of the document management and processing required in critical software systems development?
- Can the automation of the documentation-related processes benefit the software certification procedure?
- How can we optimise a Document Management Tool (DMT) in order to make it *quasi-autonomous*?
- Can a DMT output certifiable documentation independently?

In this work, we will try to answer these questions while generating valuable insights into what Agile methodologies and practices best suit critical software systems and providing tools to automate these software systems' development processes.

²<https://www.spotify.com/>

³<https://www.netflix.com/>

1.4 Objectives

This work aims to contribute to the application of Agile Methodologies in critical software systems by adopting practices and respective supporting tools to reduce time in the development of certifiable products in critical software systems development. The main focus of this work is the automation of document management and processing by creating a mechanism that streamlines the production of certifiable documentation.

The produced DMT - visible in Figure 1.1 - allows the management of the documentation through a DocOps approach. DocOps (further detailed in Section 2.4) allows the automation and the integration of the documentation development process, supporting each stage of this process, such as planning, writing, publishing, and obtaining feedback. Thus, this DMT allows the automation of the document-related processes, allowing the production of certifiable documentation that will be validated and certified afterward.

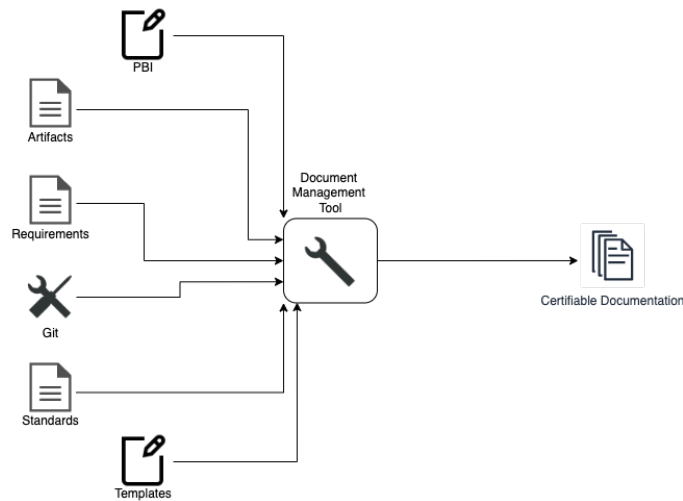


Figure 1.1: Expected DMT

In order to validate the proposed solutions, during each phase of the DMT creation (ideation, design, prototype, and implementation), there is a thorough analysis, testing, and V&V following the V-Model from CSW's professionals. The tool can also be put to practice in concrete case studies from the company.

1.5 Document Structure

This dissertation report contains eight chapters, being organized as follows:

- Chapter 1, **Introduction**, introduces the context and motivation for this work, the problem under study, and the general goals of the dissertation's project.
- Chapter 2, **Background**, scrutinizes the key concepts of this work, required to its comprehension.

- Chapter 3, **State of the Art**, describes the current state of the art of agility in critical software systems development.
- Chapter 4, **Problem Statement**, systematizes the work's problem and elaborates on the methodology and work plan to the proposed solution.
- Chapter 5, **Software Development Process Automation**, outlines the methodology and thought process behind the implementation details, based on the characterization of a DO-178C standard [36] checklist;
- Chapter 6, **Solution Proposal**, provides the tools' implementation details, as well as its limitations;
- Chapter 7, **Evaluation**, analyzes the results of the proposed tools against the original proposed implementation with a set of synthetic tests and deriving the answers to the research questions and hypothesis analysis;
- Chapter 8, **Conclusion**, summarises the developed work, discussing the contributions and the future work.

Chapter 2

Background

In this chapter, we introduce the key concepts to fully understand the work. Section 2.1 introduces the concept of Agile and its practices and methodologies, as well as some of its identified benefits. Section 2.2 describes the fundamentals of safety-critical software systems as well as their inherent characteristics and certification. The DO-178C standard is stated in Section 2.3. Lastly, Section 2.4 reviews DocOps practices.

2.1 Agile

Agile is an umbrella term that refers to a family of best-of-class iterative software development practices, methods, and techniques [44]. This concept was first introduced with the publication of the Agile manifesto [8]. It originated contrast to plan-driven models, such as Waterfall, V-model, *et cetera*, which emphasize thorough and detailed planning, design upfront, and compliance to consecutive phases of the plan [30]. The key values of the Agile manifesto are present in Figure 2.1.

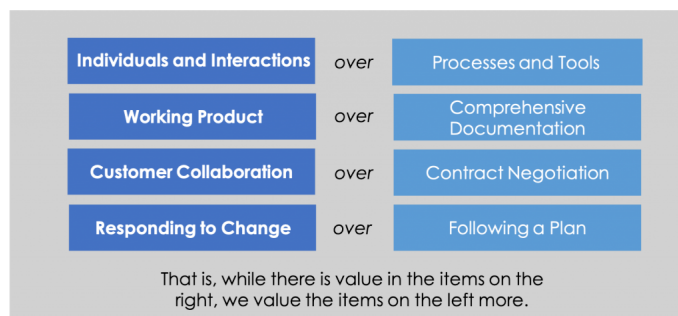


Figure 2.1: Agile manifesto values

Since Agile's introduction, its adoption has spread in different industries. According to the 1382 answers to the 2021's 15th State of Agile Survey [13] - yearly questionnaire whose main

objective is to enquire organizations and explore the agile adoption, trends, best practices, and challenges faced during the process - there was a percentage increase of 86% in Agile adoption among software development, following 2020's 37% growth.

97% of the organizations that answered the 2021's State of Agile Survey responded that they adopt Agile practices. Figure 2.2 shows how long this 97% have been using agile methods in their organizations.

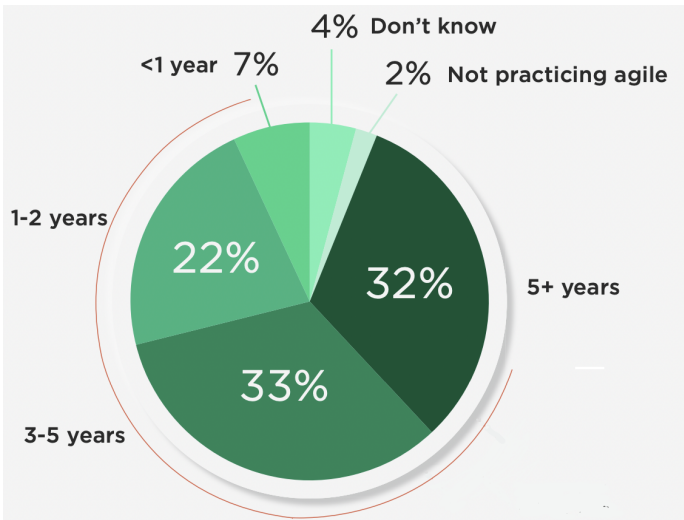


Figure 2.2: For how long Agile has been adopted, extracted from [13]

According to 2020's The State of Agile Survey [12], visible in Figure 2.3, there is a low inclusion of Agile practices in what can be considered a critical software system, assuming that transportation, healthcare, and pharmaceuticals only represent a total of 7% of the areas where Agile is adopted.

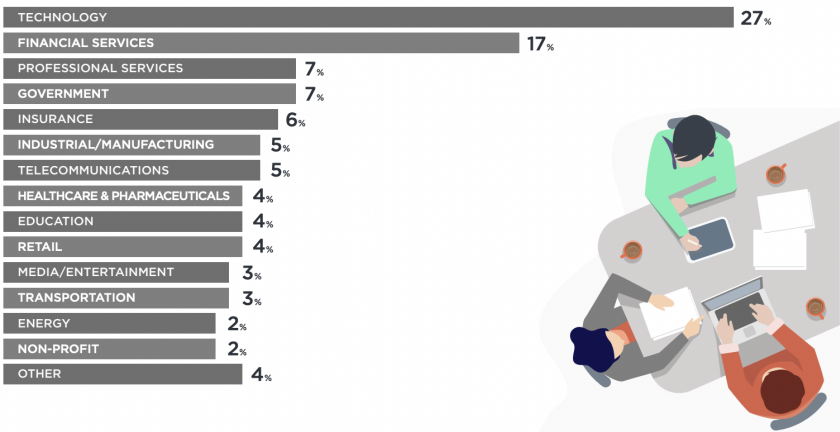


Figure 2.3: Types of industries using agile methods, extracted from [12]

It is essential to mention that Agile includes different methods and techniques, such as XP (Xtreme Programming), Scrum, *et cetera*, that can fit into each team and industry's needs.

2.2 Critical Software Systems

Critical software systems are systems whose failure may lead to significant negative consequences, such as injury or loss of life, damage to the environment, unauthorized disclosure of information or serious financial losses [20, 40]. Due to the risk involved within these software systems, some of its inherent characteristics are the fact of being extremely costly, fault-tolerant, secure, maintainable and standard compliant.

There are four types of critical software systems, as we can observe in Figure 2.4, depending on the implications caused by its failures. Given the context of this work, with the collaboration of CSW, there is one type that will be focused on this work: safety-critical software systems.

Type of Critical	Implication for Failure
Safety-Critical	May lead to loss of life, serious personal injury, or damage to the natural environment.
Mission-Critical	May lead to an inability to complete the overall system or project objectives; e.g., loss of critical infrastructure or data.
Business-Critical	May lead to significant tangible or intangible economic costs; e.g., loss of business or damage to reputation.
Security-Critical	May lead to loss of sensitive data through theft or accidental loss.

Figure 2.4: Types of Critical Software Systems, extracted from [20]

As we can observe in Figure 2.4, safety-critical systems can be considered as a type of critical software systems. A safety-critical software system can be classified as:

"A software system whose failure could result in loss of life, significant property damage, or damage to the environment." [25]

Due to the involvement of physical risks, development of safety-critical software system is typically undertaken within respect to particular generic or domain specific standards or domain specific standards or other regulatory constraints [17], imposed by the certification authorities.

Given authorities recommendations (and still not recognizing Agile methodologies as valuable), there is the settlement of plan-driven methodologies in safety-critical software development. Islam, G. *et al.* [21] states that there is pressure to use Waterfall (and other plan-driven methodologies) in this area's teams and companies.

2.3 DO-178C Standard

Entering more in detail about the aerospace domain, since this is an important safety-critical software system in the context of CSW, the DO-178C standard provide guidelines for the production of software for airborne software systems and equipment that performs its intended function with a level of confidence in safety that complies with airworthiness requirements [49].

To ensure this standard's safety and modularity, RTCA/EUROCAE joint committee has produced four complementary development technology supplements, published as stand-alone documents. Figure 2.5 shows the relationship between DO-178C standard and those supplementary documents.

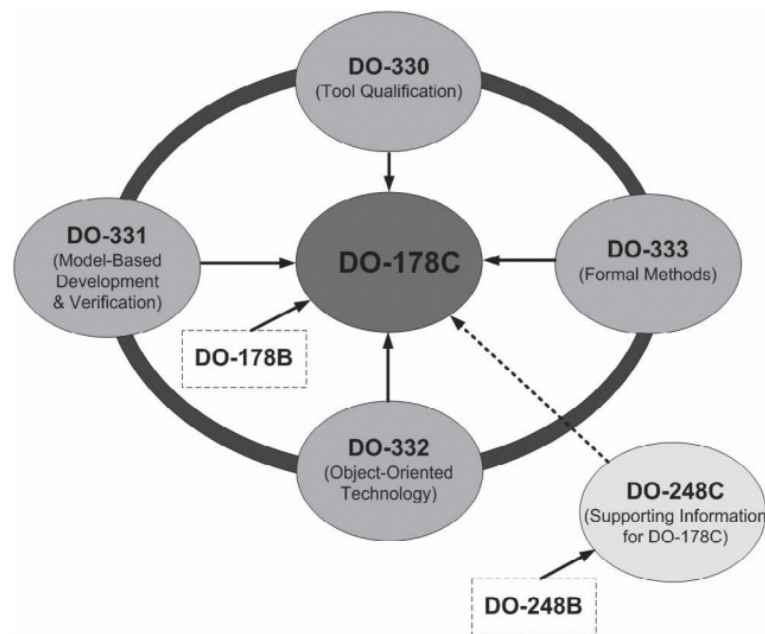


Figure 2.5: DO-178C standard and its supplementary documents, extracted from [49]

Furthermore, DO-178C standard also provides a software system safety assessment process, based on the failure conditions associated with the software component [14]. Below we have detailed each criticality level caused by software's anomalous behaviour determining or contributing to a failure of a software system function [19].

- **Level A: Catastrophic.** Failure condition resulting in multiple fatalities and loss of the aeroplane;
- **Level B: Hazardous.** May not kill everybody, but there is potential for that hazard;
- **Level C: Major.** Can cause people to become injured and the aircraft uncontrollable;
- **Level D: Minor.** May have some effect but it can be overcome by the aircraft, and pilots can maintain control;

- **Level E: No Effect.** Has no effect. While this error does not affect the functioning of the aircraft, justifications about this issue need to be provided to the competent authorities.

2.4 DocOps

DocOps is an emerging theme in software development that derives from DevOps. Atlassian [4] classifies DevOps as *"a set of practices, tools, and a cultural philosophy that automate and integrate the processes between software development and IT teams"*. Thus, the definition of DocOps, can also derive from DevOps' definition [41]:

"A set of practices that works to automate and integrate the process of developing documentation across engineering, product, support, and, of course, technical writing teams."

Figure 2.6 graphically represents the life cycle of the DocOps practices from the first phase, planning, until the analysis/feedback, that in the context of critical software systems development represents the V&V and certification process.

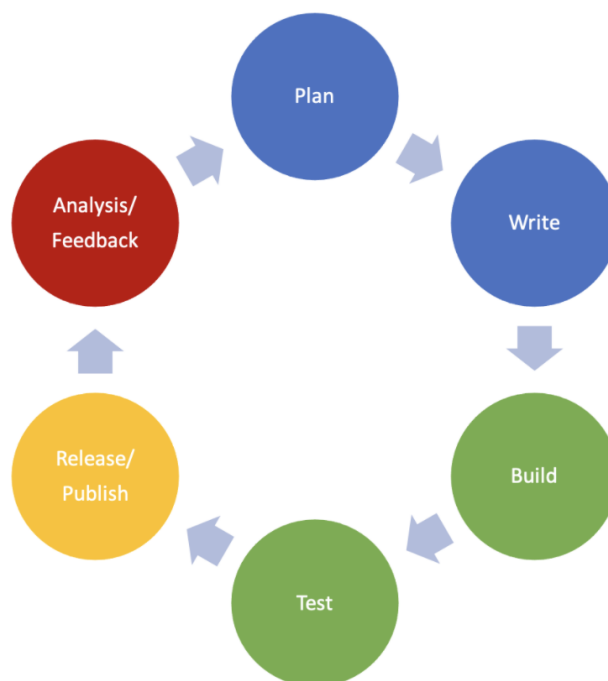


Figure 2.6: DocOps life cycle

Chapter 3

State of the Art

This chapter details the state of the art for the topics of interest to this work: (1) the adoption of Agile methods in critical software systems development, (2) document automation and management in the software development industry, and (3) DO-178C standard and documentation. The methodology is presented in Section 3.1. Section 3.2 studies the benefits and obstacles of Agile adoption in critical software systems. The current applied document management and automation practices are introduced and described in Section 3.3. As for Section 3.4, there is the presentation of the relation between DO-178C standard and documentation, including pros, cons, and requirements. Finally, Section 3.5 summarises the main findings on the aforementioned topics.

3.1 Methodology

An *ad hoc* methodology was used for the literature review. Different research questions were defined (Subsection 3.1.1) and its relevant terms were used to query different databases (Subsection 3.1.3). The results selection and its analysis iterative process is specified in Subsection 3.1.2.

3.1.1 Literature Research Questions

The following literature research questions (LRQs) were developed to guide the literature review process:

LRQ1: What are the benefits of introducing Agile methodologies to critical software development?

This work proposes the adoption of Agile methodologies within the context of critical software development. As such, a deep understanding of the benefits of these methodologies and practices is needed.

LRQ2: What are the obstacles impeding the adoption of Agile Methodologies? Can they be surpassed?

As we are adopting Agile methods, there is the need to recognize the concerns and obstacles regarding this fact. We aim to understand what these challenges are and how to mitigate them.

LRQ3: What are the current documentation automation tools and practices? Did they contribute positively for the certification of documentation?

Existing work on documentation automation is essential to this research. Analyzing the work on this subject is key to seeing where it falls short and what can be considered and reused for the context of critical software systems.

LRQ4: What are the implications of DO-178C standard in the documentation processes?

As the aviation area is relevant within CSW's context, it is necessary to recognize the impact of compliance with the DO-178C standard in document creation and management processes.

3.1.2 Process

1. Given a literature research question, design a query to fulfill its requirements;
2. Submit query into different databases;
3. Identify and select the relevant results based on title, abstract, methodology (if applicable), and conclusions;
4. Read and analyze the articles and then look up the references to identify missing concepts and topics;
5. Go back to step 2, refining the query based on previous results.

3.1.3 Databases

The search process included using digital libraries such as ACM Digital Library, IEEE Xplore, Science Direct, Scopus, and Research Gate.

3.2 Agile in Critical Software Systems

As explained before, with the increasing demand for the aerospace sector to obtain agile transformation to respond faster to the customer's needs and requirements changes, many companies and researchers started to investigate this field. Since this is a complex and challenging environment, researchers want to understand the adjacent benefits and concerns that Agile methodologies adoption would bring to the critical software development industry.

The following subsections (Subsections 3.2.1 and 3.2.2) analyze the benefits and concerns of the implementation of Agile methods in the critical software development industry, focusing mainly on the documentation and certification processes, given the goal of the work.

3.2.1 Benefits (LRQ1)

Vanderleest, Steven *et al.* conducted different pilot tests with the implementation of some Agile techniques in a critical software system development [44]. Those techniques were:

- **Test Driven Development (TDD):** A programming practice that instructs developers to write new code only if an automated test has failed and to eliminate duplication. The goal of TDD is 'clean code that works.' [22];
- **Pair Programming:** The code production is written by pairs of programmers working together at the same workstation [9];
- **Continuous Integration:** Programmers always work on the latest version of the software. Therefore they check in their code and integrate several times per day [9];
- **Iterative Approach:** Procedure to building software in which the overall life cycle is composed of several iterations in sequence. Each iteration is a self-contained mini-project composed of activities such as requirements analysis, design, programming, and test. The goal for the end of an iteration is an iteration release, a stable, integrated, and tested partially complete software system [27];
- **Fixed Length Iterations:** Short iterations with a fixed length, providing increments of shippable and usable products, which usually allow customer feedback.
- **Client-Driven, Adaptive Planning:** Agile process that requires a more consistent involvement from the client during all phases of the program, ensuring that the client is aware of and approves all of the decisions and changes of direction in the program, allowing a more direct impact in the development process.

The authors mention that these techniques helped the aerospace software development teams to deal with some difficulties, despite the specificity of the DO-178B standard. However, the authors don't go into detail on what was the concrete approach to the inclusion of these techniques and what were the produced results and even conclude that "*larger-scale demonstrations are now needed*".

Kasauli, R. *et al.* conducted a systematic literature review that resulted in the discovery of several benefits associated with the utilization of Agile practices in the critical software systems development [24]. The authors elaborated a list with the key benefits that were most discussed in the literature as presented in Table 3.1.

Aiming towards the creation of a process for the adoption of Agile methodologies, A. Sidky and J. Arthur proposed a three-phased procedure [38]:

- **Go-NoGo Phase:** Looking for the success factors of each practice and checking their utility;
- **Discarding Inappropriate Practices Phase:** Identify conflicts between the chosen Agile practices and the system;

Table 3.1: Key benefits of Agile practices adoption, adapted from [24]

Benefit	Number of Article References
Improved stakeholder involvement	5
Reduced costs	5
Improved quality	4
Efficient Use of available information	3
Improved safety culture	3
Improved opportunities for reuse	3
Improved management of changing requirements	3
Improved prioritisation	2
Mapping of functional and safety requirements	2
Better test cases	2

- **Determining Practices to Adopt Phase:** Identify organizational objectives and dependence between Agile practices.

This method allows organizations that develop critical software systems to take advantage of the benefits of Agile practices, ensuring that these are suitable for the organization, achieve its objectives and do not conflict with the software system’s characteristics or impact the system’s qualities. The authors also pointed some of the observable Agile benefits such as *early return on investment*, *improved quality*, *better team morale*, and *enhanced client relationships*.

Vuori M. also states that Agile methods have been expected to provide more control for the development process and to be able to deliver value to customers and developers earlier. The author also considers Agile methods to allow meeting challenges in changing requirements more efficiently than previous process life cycle models [45].

3.2.2 Concerns (LRQ2)

Tordrup L. and Nielsen P., conducting interviews with critical software development teams, pointed four different challenges to the implementation of Agile practices within this context [43]:

- **Use of documentation:** Keeping documentation to a minimum was problematic since it had to be standard-compliant;
- **Requirements engineering:** Dealing with uncertainty in requirements had a negative impact on the developers;
- **Life cycle:** Given the fact that critical software development needs coordination with the produced hardware [35], due to its embedded development, the software development team’s sprints were often interrupted by hardware fabricating teams, prejudicing the development process;
- **Testing:** Adopting an iterative testing strategy requires significant changes in the work practice, which constituted a challenge to developers. The need to learn new Agile testing skills was also problematic.

Islam, G. and Storer, T. conducted an interview process and posterior V&V by executing pilot tests collaborating with a company. As a result, the authors were able to create a list of fifteen concerns, grouped into four topics [21]:

- **Pressure for the Waterfall:**

- Agile software development advocates incremental design, but safety standards require upfront design as a necessary input for hazard analysis;
- Regulatory standards are perceived as mandating Waterfall and not permitting agile methods;
- The prevalence of fixed-price contracts for pre-agreed requirements in critical software systems projects is not readily compatible with agile software development;
- The actual time taken to complete the tasks always turns out to be more than it is estimated in the beginning, particularly due to integration complexity in critical software systems projects;
- Requirements are difficult to modularise in critical software projects because the functionalities are so interdependent that it is very hard to separate them.

- **Coordination amongst Stakeholders:**

- Software teams lose visibility during the integration phase. Agile methods lack guidance on integration with hardware;

- There is a complex network of customers that obstructs agile ceremonies such as the Sprint Review;
- Software, hardware, firmware, and other teams in critical software systems work function independently according to their own schedule causing plans to become mismatched.

- **Documentation and Communication:**

- Face-to-face informal contacts dominate communication, causing project related information to be lost;
- Frequent releases increase over-heads and costs because they must be accompanied by supplemental documentation to achieve certification.

- **Cultural Challenges:**

- The Software team has no practical example to follow for applying agile methods, and they lack the resources to experiment;
- The teams need guidance on how to scale agile methods for use in a large multi-team context;
- The organizational mindset require convincing about the benefits of agile software development.

When considering the documentation-related obstacle, *"frequent releases increase over-heads and costs, because they must be accompanied by supplemental documentation to achieve certification"*, the authors suggest the automation of the documentation writing and processing.

Wils, A. *et al.* divide the development of critical software in three different phases: *software phase*, *embedded phase*, and *certification phase*. The first of the three phases, as it indicates, comprises independent software development. According to the authors, this is where Agile methods can be applied. During the *embedded phase*, the software must be deployed and tested on the target hardware. As for the last stage, *certification phase*, there is the reviewing of the documentation and the software, including its V&V, and posterior certification by regulatory entities [47].

Despite recognizing some Agile practices as functional and applicable, mainly in the *software phase*, and likely to be implemented in this context, the authors consider that the full adoption of Agile methods is difficult, mainly because of the *certification phase*, documentation management, and the regulatory authorities that do not recognize Agile practices as useful.

Conducting a pilot study in a high-integrity software system development, Paige, R. *et al.* encountered six major challenges to overcome: *communication*, *documentation*, *customer participation*, *multiple-domain engineering*, *testing* and *incrementality* [31].

Vuori, M. claims that the documentation might be left out when adopting Agile methodologies in critical software development given the complexity of the writing process and posterior certification [45].

In his systematic literature review, Kasauli, R. *et al.* pointed out several concerns associated with the utilization of Agile practices in the critical software systems development [24]. The authors elaborated a list with the key concerns that were most discussed in the literature as presented in Table 3.2.

Table 3.2: Key concerns of Agile practices adoption, adapted from [24]

Concern	Number of Article References
Strong focus on documentation	9
Upfront planning	9
Safety standards to facilitate a waterfall mindset	6
Lack of trust in agile methods	5
Flexibility vs Safety	4
Difficult to manage knowledge flow between stakeholders	4

The above-mentioned gathered information is summarized in Table 3.3.

Table 3.3: Concerns to Agile implementation in critical software systems software development

	Number of Concerns	Certification as a Concern	Obtained By
Tordrup, L. <i>et al.</i>	4	Yes	Interviews
Islam, G. <i>et al.</i>	13	Yes	Interviews and pilot tests
Wils, A. <i>et al.</i>	N/A	Yes	Literature review
Vuori, M.	4	Yes	Literature review
Paige, R. <i>et al.</i>	6	Yes	Pilot study
Kasauli, R. <i>et al.</i>	6	No	Literature review

Despite the lack of empirical evidence that support all of the obstacles pointed in these works, there is a clear notion that, apart from other concerns, the document management and, consequently, the certification process of critical software generate preoccupation for the Agile methodologies' adoption.

3.3 Documentation Automation (LRQ3)

Achachlouei, M. *et al.* [3] points Jupyter Notebook¹ as a potential candidate as a documentation system. The authors consider that due to its programmatic nature, there is the offer of different advantages in a Document Automation workflow such as:

- Ability to retrieve information from external databases;
- Ability to add more automation features using Python libraries which offer graphical user interface control so the users can decide how their documents should be composed or modified, making the documentation process easier;
- Ability to offer advanced features, such as collaboration and authentication;
- Support for enterprise scalability;
- Support for a wide variety of programmatic languages that can work with notebook's underlying standard format, i.e. JSON. Working with this format in the notebook-based Document Automation systems is more convenient than working with XML, which is common in many Document Automation systems;
- Text can be added uniformly in Markdown and converted to the final document format (Latex/PDF/HTML/DOCX).

The authors consider, as the greatest advantage, Jupyter's open-source nature and its use of JSON as the notebook document format.

Furthermore, the authors provide an architecture for model-based systems engineering documentation workflows, visible in Figure 3.1. It is important to mention that, despite the vast majority of critical software systems fitting in this type of system, not all of them follow this standard.

Comoretto, G. *et al.* followed a model-based systems engineering approach to verify and validate Rubin Observatory's software and utilized Jira² and "DocSteady", a tool developed to generate LaTeX test documents [10], that would extract information from Jira. This allowed a reduced time to produce V&V documents, better integration with the project's software system engineering model, and full traceability of system requirements.

When it comes to the automatic generation of documents to flight software, Lee, Jae-Seung *et al.* state Doxygen³ as a useful tool [28]. Given that this tool allows the generation of documents from source code, the authors consider that it significantly reduces the time and complexity of the development process.

¹<https://jupyter.org/>

²<https://www.atlassian.com/software/jira>

³<https://www.doxygen.nl/>

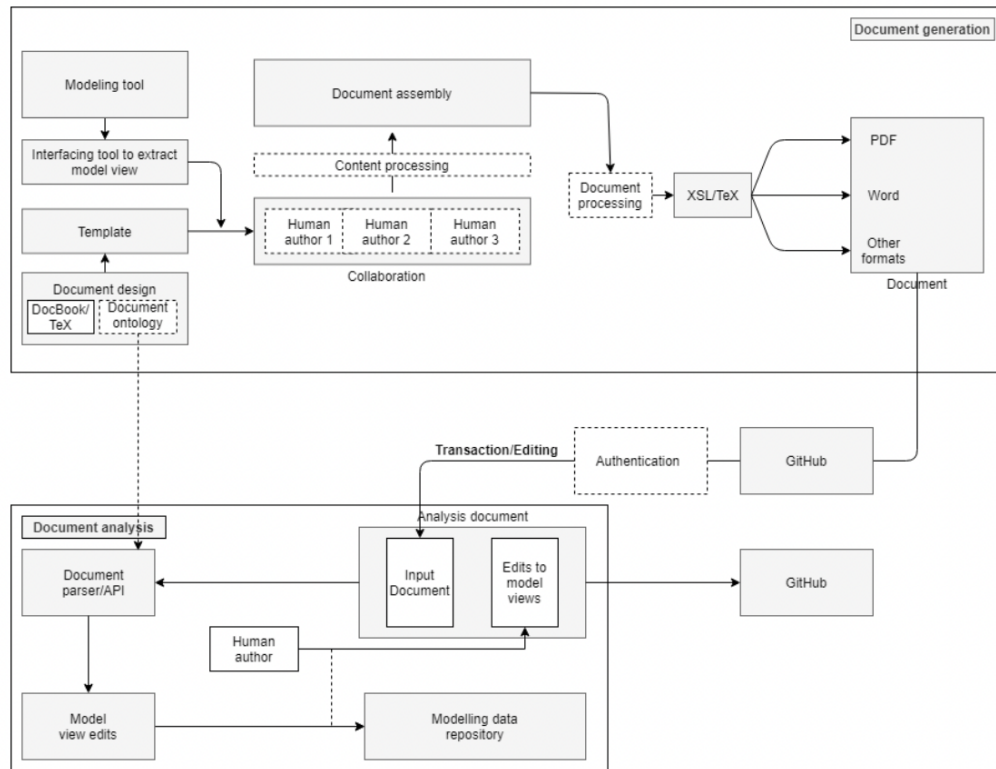


Figure 3.1: Architecture for model-based systems engineering documentation workflows, extracted from [3]

Aguiar, A. mentions a set of tools to support the documentation process named XSDoc. This infrastructure supports source code written in Java and C++ programming languages, models described in UML, and simple integration with Eclipse IDE⁴ [2]. According to the author, this combination of technologies helps reduce the effort required to document frameworks, making documentation easier to integrate into the software development environment. The edition, revision, and evolution of the documentation contents are also facilitated in a controlled and structured way.

Forward, A. and Lethbridge, T. conducted a set of interviews with developers, where participants support the idea of improving document automation, although more research is required to determine what data is required and should be extracted and by what means [15]. The author also gathered a list of what documentation processes and tools should focus on:

- Focus on content and allow the author to easily create and maintain content-rich documents;
- Focus on availability and allow for comprehensive publishing capabilities;
- Focus on examples and allow for better integration of examples within a document.

⁴<https://www.eclipse.org/>

Lankester, R. states that document automation "*can be of significant value to the client and can be used to generate income, build stronger relationships and win future work.*" [26]. Furthermore, the author provides a list of benefits adjacent to document automation:

- Efficiency;
- Reducing risk;
- Improving document quality;
- Reducing turn-around times for clients;
- Knowledge management and development benefits.

3.4 DO-178C Standard and Documentation (LRQ4)

As mentioned in Section 2.3, DO-178C standard provides guidelines for the production of software for airborne system, including the certification process.

Youn, W.K., *et al.* stated that DO-178C standard requires rigorously iterative processes and extensive documentation throughout the entire software life cycle, a fact that contributes to the high expense of commercial aviation systems [49].

Hilderman, V. describes the most commonly obtained benefits from DO-178C standard based on his 25 years of experience [18]. Documentation has an impact in two of them:

- The precise documentation of Parameter Data Items, unused software structures, and high and low-level requirements provides more thorough testing through the verifying under the DO-178C standard;
- There is improved reusability via the rigorous and consistent documentation, modularization, enforcement of documented modern engineering principles, and reviews required by DO-178C standard. By ensuring all the above-mentioned techniques and processes, reusability - stated by the author as a Holy Grail in software - is greatly improved.

Linling, S. *et al.* gathered a list of the required documents for the certification process of a DO-178C standard compliant software system [29]. This list, visible in Figure 3.2, demonstrates the significant amount of documentation produced during the safety-critical software system development that will be certified by regulatory authorities.

3.5 Summary

Section 3.1 describes the methodology used to perform the literature review on the topics of this work. In this section, literature research questions are presented, the databases used are enumerated, and the iterative process to filter relevant literature documents and to answer the research

Certification Data	
Certification Plan	Preliminary Aircraft / System Safety Assessment
Development Plan	Aircraft / System Safety Assessment
Design Description	Common Cause Analysis
Validation Plan	Validation Data
Verification Plan	Verification Data
Configuration Management Plan	Evidence of Configuration Management
Process Assurance Plan	Evidence of Process Assurance
Configuration Index	Certification Summary / Compliance Report
Functional Hazard Assessment	

Figure 3.2: DO-178C standard required certification data, extracted from [29]

questions are presented. Section 3.2 is divided in two different subsections and focuses in **LRQ1** and **LRQ2**. Subsection 3.2.1 focuses on the presentation of the numerous benefits that Agile adoption brings to the development of critical software. As for Subsection 3.2.2, there is a listing of different concerns about the inclusion of Agile in this industry, focusing particularly on the documentation and certification processes. There is a clear idea that documentation and certification raise preoccupation for Agile adoption. Despite these concerns and obstacles, there is still a lack of empirical proof that sustains all of them, creating a sense of opportunity for Agile inclusion.

Section 3.3, which focuses on **LRQ3**, demonstrates the state of the art of document automation in the software development industry. There is the presentation of different benefits and advantages of document automation, as well as the tools that currently focus on that purpose and contribute with valuable insights to the work of this dissertation.

Despite the existing automation tools in different industries, there is a lack of a DMT that comprises all the critical standards and requisites for certification documentation. Nevertheless, the actual contributions in document automation are relevant and provide a constructive contribution to our work.

Section 3.4, focusing on **LRQ4**, describes the benefits, concerns, and the imposed needs by the aviation DO-178C standard within the documentation and certification processes in safety-critical software development. This information is of greater importance, given our work's proposed DMT and its goal of automating document-related processes. Nevertheless, the certification process, which includes documentation V&V, is lightly discussed in the publications. The research papers discuss the complexity, but there is little information about what is needed.

Based on the information found in this literature review, in the following chapter, we describe the problem under study in depth.

Chapter 4

Problem Statement

This chapter describes the problems related to this work. Specifically, Section 4.1 states the current issues found during our research that will be tackled in this dissertation. Section 4.2 identifies the scope and focus of this work. Then, Section 4.3 defines the set of requirements we intend to address in this work. Section 4.4 elaborates on the hypothesis and research questions of the dissertation.

4.1 Current Issues

In Chapter 3 we present the current state of the art on a variety of topics related to Agile adoption in critical software systems and document automation in the software industry and concretely in aviation software development. Despite the benefits and successful cases, there are still open challenges and issues within these topics.

Firstly, the critical software industry doesn't yet consider Agile methodologies and techniques as valuable and fruitful when compared to plan-driven development. This leads to cultural challenges since these methodologies are too implanted in the industry. Furthermore, Agile practices lead to light documentation, which is the opposite of what this industry needs: documentation is pivotal for the V&V and certification processes. Documentation is also a big concern when referring to Agile. There is a strong focus on documentation, divergent from Agile.

When it comes to document automation, despite the existence of several tools, none of them automates the process with the output of certifiable documents. Furthermore, those tools don't comprise all document-related processes, unlike our proposed DMT. In addition, the needs of the aviation industry, with special emphasis on the DO-178C standard, are not included in the current documentation automation tools.

In addition to the previously mentioned issues, there is also a lack of research on Agile inclusion in critical software development and documentation automation within this context.

4.2 Scope

Based on the literature shortcomings and current issues identified in the previous subsection, this dissertation aims to study different Agile practices and methodologies that can be adopted within the development of critical software systems, making the process more efficient and flexible to requirements change.

Additionally, the scope and main objective of this work is to design, prototype and implement a DMT that allows the automation of documentation management in critical software systems. This is to be developed following DocOps and Agile practices that still have few adoption in the industry.

4.3 Requirements

As discussed in the previous section, this work includes the implementation of a DMT that is responsible for the document-related processes within the critical software development context. Such implementation should satisfy the following requirements:

- **R1: Manage the initial documentation requirements**, so that the DMT output documentation follows the standards applied to the critical software system.
- **R2: Provide easiness in document modification**, allowing the needed changeability of documentation through all phases of the software development.
- **R3: Make the document creation and modification processes *quasi-autonomous***, reducing the intervention of developers in the documentation process. This allows a diminished risk of human-flaw, improved version control and less time spent with an auxiliary process.
- **R4: Deployment of certifiable documentation**, that is, automatically generate the needed documents for certification from regulatory entities.

4.4 Hypothesis & Research Questions

Following the requirements, we claim the following hypothesis:

"Document automation through a DMT within the context of critical software systems can improve the software development process, achieving greater efficiency, quality, and effectiveness of the document and certification-related processes."

To validate this hypothesis and to achieve the proposed goals, the following research questions (RQs) were identified to guide this work:

RQ1: How can Agile practices help to improve the efficiency and effectiveness of document management and processing required in critical software systems development?

Critical software systems' teams remain reticent in adopting Agile practices to its software development processes. Given its proven value in different industries, it is vital to understand

the impact of Agile practices in this context, concretely in document management and processing procedures.

RQ2: Can the automation of the documentation-related processes benefit the software certification procedure?

The automation of document-related processes has improved the efficiency of the software development operations in different areas and industries. Can this automation be applied to critical software documentation?

RQ3: How can we optimize a DMT to make it *quasi-autonomous*?

Reducing the workload from critical software developers in document-related processes is not a trivial task. Thus, it is essential to identify to what level of extent a DMT can be autonomous in dealing with these processes.

RQ4: Can a DMT output certifiable documentation independently?

Regulatory authorities verify and certificate, according to thorough standards, critical software before it being accepted. Therefore, there is the need to know how independent can a DMT be in the process of providing faultless outputs.

4.5 Methodology

CSW collaboration is pivotal for accomplishing the objective of a DMT for documentation management and processing automation. The examination of the DO-178C standard checklist allows a better understanding of the needs of the safety-critical software industry, as well as the co- design and iterative V&V in the concrete context of projects from the company. Given this collaboration, professionals thoroughly analyze the produced results and put them into practice in concrete case studies, validating whether the proposed software system fulfills the requirements. Furthermore, the goal is also to interview different critical software development companies to get a broader V&V and testing of the generated mechanisms and tools to automate the development process.

Chapter 5

Software Development Process Automation

In this chapter, we explore the process behind the ideation of the software to be implemented in this work. Given the context specified in Section 5.1, focusing on aviation software development, there is the need to analyze every requirement to create certifiable documentation. Section 5.2 presents the methodology used to characterize each above-mentioned requirement, determining the automation type (if possible) that can be used, with the results being presented in Section 5.3. Finally, Section 5.4 summarises the information of the aforementioned topics.

5.1 Context

The development of critical software, as mentioned before, complies with multiple standards and regulations defined by certification authorities. These requirements are delivered to the software development company before the implementation and will be checked upon each software delivery and corresponding documentation [48].

Given the aviation-related projects of CSW [39], the set of verification processes that the regulatory authority will make needs to be ensured during the development and upon delivery of both software and documentation. This list, according to the DO-178C standard, can be verified in Appendix A.

According to Appendix A, there is a total of 393 verifiable aspects within the software development process. To better understand the automation of this process, a thorough classification of each of these aspects needed to be made. This classification procedure allows the ideation and prototyping of executable scripts to reduce software developers' workload and automate the V&V processes, contributing to the most accurate releases and deliveries approach.

5.2 Approach

To organize the requirements checklist into its automation type, we needed to define criteria, given the requirements type, phase of the development, and needed developer contribution. The following list of automation types was defined:

- Manual;
- Software;
- Pipeline/Tests;
- Scripting.

5.2.1 Manual Automation

The development of a critical software system follows a Software Development Life Cycle (SDLC) according to the project management model/methodology implemented. Predominantly, these software systems follow a Waterfall approach, whose SDLC is depicted in Figure 5.1.

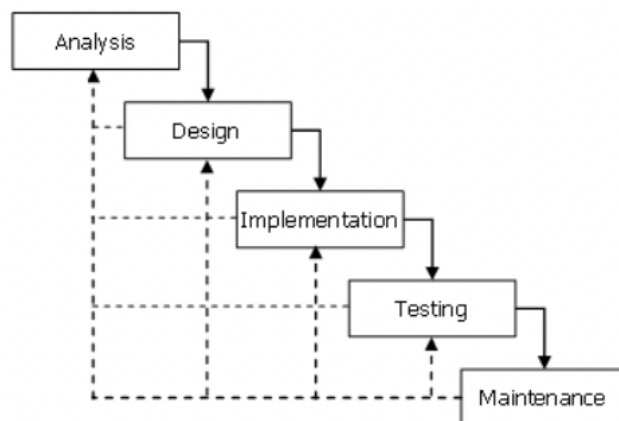


Figure 5.1: Waterfall Software Development Life Cycle, extracted from [7]

The first two phases of the Waterfall SDLC, Analysis and Design, comprise the planning, quality assurance, product feasibility, risk management, requirements engineering, architecture, and product design [42]. These operations undoubtedly require manual interaction, from requirements negotiation, understanding, and negotiation to the product's ideation, design, and prototyping.

Concretely, in the aviation documentation context of CSW, manual contact is needed with a big part of the information that will be present in the documents, i.e., naming convention, low-level and high-level requirements, standards to be followed, *et cetera*.

Thus, this automation type defines all of the human processes that can be automated without the implementation of any software by improving stakeholder communication and decision-making, standardizing naming conventions, plan, design, software, and testing practices, among other automatable processes.

5.2.2 Software Automation

The following phase of the SDLC concerns the implementation of the software. Therefore, some certifiable aspects of critical software development can be automated.

Given the DO-178C standard checklist, software development needs particular concern with standards and methodologies, tools, parameter management, compiling, planning, inputs, availability, robustness, *et cetera*. These software aspects can be automated through human decisions (part of Subsection 5.2.1 automation type) or software mechanisms.

The leading observable software automation that could be performed in the critical software development concerns the tools considered in the development, the reusability of the developed software, and the introduction of good design patterns and Machine Learning (ML) algorithms, able to reinvent themselves and computerize the software development process [37].

5.2.3 Pipeline/Tests Automation

Test Automation is one of the best ways to increase the effectiveness, test coverage, and execution speed in software testing. The major perceived benefits of testing automation include quality improvement through better test coverage and that more testing can be done in less time [23]. Despite the good results of test automation, human involvement is still needed in the selection of test cases.

Paleska [32] considers that the main effort should be invested in functional testing, having structural tests designed and executed to cover the remaining system structure not reached by preceding tests. This consideration is based on the following factors:

- The most severe design and implementation errors are caused by erroneous specifications or their incorrect interpretation. Therefore tests comparing the implementation behavior to the specification are likely to detect the most critical errors.
- While structural tests cannot be created before the completion of the design phase, functional tests can be derived as soon as the specification is ready. They can also be used to validate the specification and test the design before entering the implementation phase.
- The specification contains all the requirements relevant for the user and - in case of dependability requirements for a critical software system - the certification authorities. Therefore the acceptance test was based on the specification

Pipeline optimization and automation can also have a great contribution to the certification, V&V of the critical software documentation. A DocOps approach of a CI/CD pipeline, as represented in Figure 5.2 allows a better testing environment, version control of both code files and

documents, allowing traceability and an easier mechanism to deploy the software, documentation patch, and relevant logs created during testing.

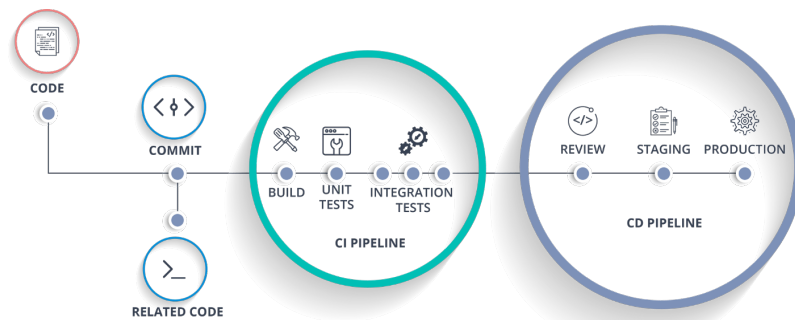


Figure 5.2: CI/CD Pipeline, extracted from [6]

5.2.4 Scripting Automation

Scripting Automation can save the costs for the overall software testing automation process, improve the speed of testing, shorten the product's launch cycle, and it can achieve an amount of work that manual tests are impossible to finish [16].

Although this categorization could be included in Subsection 5.2.3, its separation is relevant for the purpose of this work. Given the secrecy and confidentiality of a critical software project, the work produced during this dissertation should first be a part of an independent environment, acting as unattached scripts that later can be included in the Pipeline or Testing Phase of the software development project, as depicted in Figure 5.3.

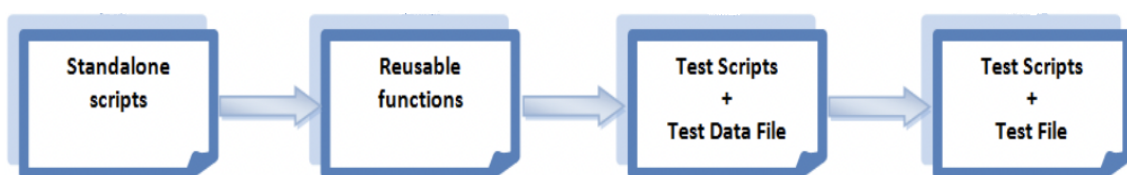


Figure 5.3: Scripting Automation evolution into Pipeline/Tests Automation, adapted from [16]

The overall purpose of Scripting Automation in this project is to create different scripts that receive various inputs (such as source code, requirements, standards, *et cetera*) and act in pre-created or to-create files composing the certifiable documentation patch, providing vital information to the software developer through logging or checklist creation.

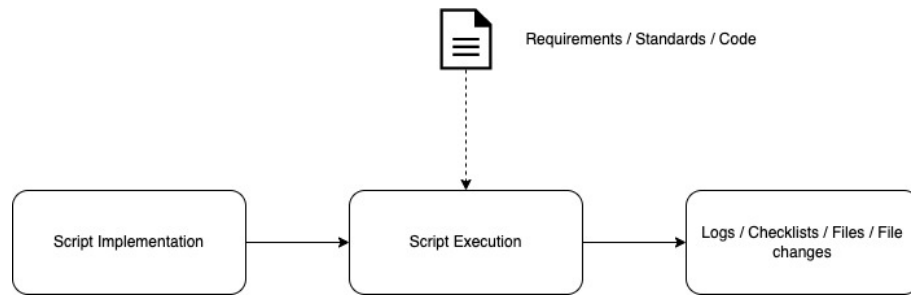


Figure 5.4: Software Automation in this dissertation’s context, adapted from [16]

5.3 Automation Analysis

As mentioned in Section 5.2, the classification of each automation type was done following the aforementioned automation types. Given the list of certifiable requirements and aspects, the results of the classification are displayed in Table 5.1.

Table 5.1: Characterization of the DO-178C standard checklist automation types

Automation Type	Number of Automatable Processes
Manual	318
Software	108
Pipeline/Tests	45
Scripting	89

As observable, the total number of Automatable Processes in Table 5.1 is greater than the total number of certifiable processes in the DO-178C standard checklist. This is due to the fact that, depending on the analyzed requirement, there might be different automation types that can be applied.

Subsequent to this classification process, there was the need of CSW’s V&V in order to begin the ideation and prototyping of tools with a positive contribution to the automation of the software development process. Considering the purpose of this dissertation’s work, there was a special emphasis and focus on the Scripting automation type, specifically addressed to documentation-related processes automation.

5.4 Summary

This chapter describes the process of classification of the DO-178C standard checklist items that precede the procedures of discussion, ideation, and prototyping of document automation tools that will form the DMT.

In Section 5.1, both context behind the certifiable requirements classification is described, and the importance of this process are described. Section 5.2 reports the methodology used for the classification process, as well as the different categories that can be allocated to each requirement.

Subsection 5.2.1 focuses on the definition and exemplification of Manual Automation techniques that can be applied to software development. As for Subsection 5.2.2, there is the description of Software Automation in the development of a critical software system. Subsection 5.2.3 characterizes Pipeline/Tests Automation, describing its contribution to software development projects and, most concretely, in the context of document automation. This Subsection is also co-related with Subsection 5.2.4 since the first can be considered as a progressive and more structured phase of the latter. This Subsection introduces Scripting Automation, the main focus of this dissertation's software implementation.

Section 5.3 presents an overview of the results of the classification process that are pivotal for the Solution Proposal approach in the following chapter.

Chapter 6

Solution Proposal

In this chapter, we thoroughly describe the problem's proposed solution. Section 6.1 outlines the methodology followed to find which executable scripts to implement.

The details of the SCIR Generation Tool are presented in-depth in Section 6.2 scrutinizing CSW's Process Overview in Subsection 6.1. As for Subsection 6.2.2, the implementation details are dissected.

Section 6.3 analyzes the implementation of the Document Traceability Automation Tool, including its pseudocode and flowchart.

The limitations of the produced work are described in Section 6.4, while the comparison between the results and the initially proposed DMT is made in Section 6.5.

Finally, Section 6.6 summarises the aforementioned topics.

6.1 Introduction

Given the results obtained in the last Chapter (more concretely in Section 5.3), we observe that scripting development and execution can be a valuable asset to achieve 89 certifiable requirements of the DO1-178C standard. Together with Critical Software, there was a discussion of possible tools that could be implemented and their overall contribution to the software and product documentation process.

To better fit the current needs of CSW critical software projects, the analysis of the classification results led to the ideation of a SCIR creation executable script. Posterior executable scripts that could lead to the automation of the documentation-related processes were assigned a low priority, depending on the length of the above-mentioned SCIR creation script.

6.2 Software Configuration Index Report Automation

The Software Configuration Index (SCI) identifies the configuration of the software product, containing one configuration item or a set of configuration items [1]. Its report is, therefore, a valuable artifact for the software development guidelines and posterior certification process.

The SCIR can be created through files and logs in a version/revision control tool. This report will encapsulate the artifacts produced during the different software development process overview stages, as described in the following Subsection (6.2.1), providing information about each document's stage, version, location, ID, and responsible developer.

6.2.1 Critical Software Process Overview

The current software development process adopted in projects by CSW contains four different stages (each one explained in Sub-subsections 6.2.1, 6.2.1, 6.2.1 and 6.2.1):

- Specification Stage;
- Software Implementation Stage;
- Software Testing Stage;
- HSI & System Testing Stage.

Prior to the beginning of these above-mentioned stages, there is the planning of the Product Backlog Items (PBI) to be included during each iteration of the software development process. The Work Management Tool used, Jira [5], contains all of the PBI, requirements, and tasks related to each feature to be implemented.

Each of the stages starts when all the artifacts from the previous stage are prepared, following a similar set of activities. After the final event of the last stage, the artifacts should be generated, creating a certifiable package that will be released into certification by competent authorities.

Figure 6.1 helps understand the above-explained process overview adopted by CSW.

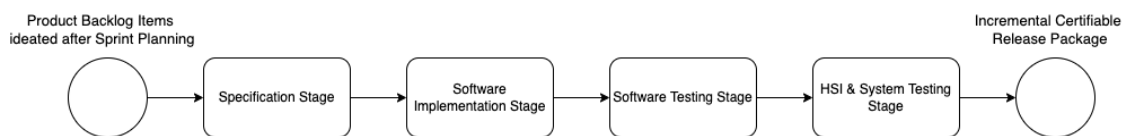


Figure 6.1: The software development process under development within a PhD research [33]

Specification Stage

During the Specification Stage, the software system specification and design are described and reviewed.

At the end of this stage, a new documentation package can be built with all the artifacts produced, but the package is not certifiable.

Software Implementation Stage

The Software Implementation Stage marks the start of the software development, starting by specifying High and Low Level Requirements, Review and Software Implementation.

After this stage's end, a new documentation package is built with the joint artifacts of both this phase and the Specification Stage, but the package is not certifiable yet.

Software Testing Stage

During the Software Testing Stage, one important stage of the process, Unit Tests and Integration Tests are enhanced and worked on. This stage marks the phase of testing and integration of the software in the main branch of the Version Control System.

Posterior to this stage, the produced data package is not certifiable and comprises the artifacts of the prior finished stages.

HSI & System Testing Stage

The last stage, HSI (Hardware & Software Integration) and System Testing, deal with the testing made in emulators (simulating, in this context, an airship), reproducing a real-life scenario and environment. This is the last stage ahead of the delivery of certifiable software and document, and therefore, upon its end, there is a certifiable data package containing artifacts produced in all of the stages. This data package is subsequently sent to certification identities.

6.2.2 Implementation

In order to fulfill CSW's technological requirements (following the implemented tech stack), there was the need to incorporate in the produced software the following technologies:

- Jira ¹;
- Confluence ²;
- BitBucket ³.

Given current automation projects being already studied, in the context of a PhD research [33] within collaboration with CSW, any implementation of a SCIR Generation Tool would create conflict - this topic is approached in Section 6.4 - creating the necessity of a more broad software, capable of being introduced on the critical software development industry. This fact led to a different chosen tech stack, including:

- Github ⁴;

¹<https://www.atlassian.com/software/jira>

²<https://www.atlassian.com/software/confluence>

³<https://www.bitbucket.org/>

⁴<https://github.com/>

- Python.

To fully automate the process of the SCIR document generation, without developer interaction, we can divide the process into different stages:

- Data Fetching Stage;
- Data Parsing Stage;
- Document Generation Stage.

The designed architecture of the executable script is visible in Figure 6.2.

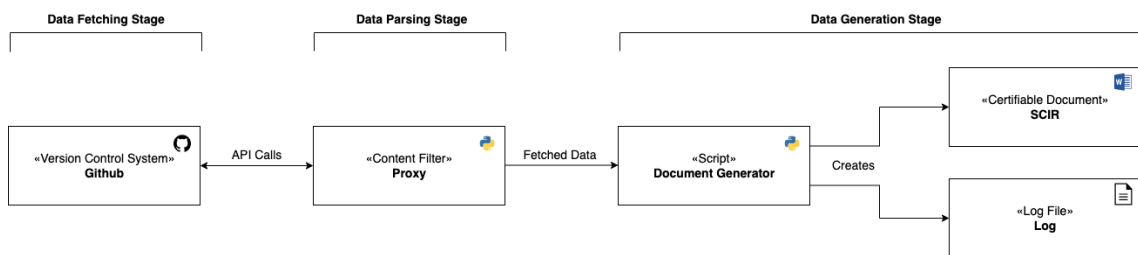


Figure 6.2: SCIR Generator architecture

The first stage, Data Fetching Stage, concerns the connection and authentication in the Github API, requesting important information from its endpoints. In order to achieve this goal, the work can be divided into two different steps. Firstly, an authentication mechanism with the Github API was implemented. This method used a Github Personal Access Token, previously configured and generated in Github Personal Area. The configuration of this Token, to allow a faultless operation of the SCIR Generation Tool, needs allowance to interact within the repositories scope. Failing this requirement will lead to the impossibility of fetching the relevant information and artifacts from Github.

A proxy-like architecture was afterward implemented to serve as an intermediary between Github API and the Document Generator script. This content filter was responsible for connecting with the required Github API endpoints. These endpoints include:

- | | |
|-----------------------|----------------------|
| • Pull Requests; | • Actions; |
| • Pull Request Files; | • Actions Artifacts; |
| • Issues; | • Artifacts; |
| • Commits; | • Users; |
| • Events; | • Repositories. |

After fetching the data and before sending it to the Document Generation Stage begins the Data Parsing Stage. This phase consists of the processing of the fetched data from the Github API. After each request is made to the API, a JSON response is received and processed.

Taking into account the variety of endpoints and different information inherent to each one, data received from each request needs different processing. Firstly, if the response has an erroneous status code, information about the error is appended to a log file that, at the end of the script execution, will provide valuable information about the SCIR generation process. Otherwise, if the response is empty, meaning that the request was successful despite not containing any information, there is also appended a warning message to the same log file. Apart from these situations, when valuable data is fetched, processing occurs, selecting the needed data for the SCIR generation, as shown in Table 6.1.

Table 6.1: Gathered data from each endpoint after parsing process

Endpoint	JSON Data After Parsing
Pull Requests	["id", "number", "title", "url", "author", "commit_sha", "state", "labels"]
Issues	["id", "state", "number", "title", "author", "pull_request"]
Commits	["sha", "author", "message", "date"]
Events	["id", "type", "author", "head", "before"]
Actions	
Actions Artifacts	
Artifacts	["id", "name", "url"]
Pull Request Files	
Users	["username", "name", "email"]
Repositories	["name", "url"]

The last stage, Document Generation Stage, begins after the processing of the received information from Github API. The proxy sends this information to a document generation script responsible for building the SCIR. The SCIR is divided into six different sections:

- General Information;
- Warnings & Errors;
- Specification Stage Artifacts;
- Software Implementation Stage Artifacts;
- Software Testing Stage Artifacts;
- HSI & System Testing Stage Artifacts;
- Miscellaneous Artifacts.

The document's general information contains the id of the report, its creation date, the number of errors and warnings. The warnings and errors section specifies these errors and warnings. The main difference between these and the ones added to the log file is the stage in which they occur and the causes. While the log file deals with warnings and errors in the data fetching and parsing stages, the errors and warnings displayed in the SCIR deal with flaws produced during this last stage (i.e., artifacts that do not belong to any process overview stage, erroneous naming).

The following sections of the SCIR contain the relevant information about the artifacts present in the repository. Each of the sections contains the information on the correspondent process overview stage. The last one, Miscellaneous Artifacts, includes the relevant artifacts that are not classified with any of the other stages.

The pseudocode in Algorithm 1 shows the first two stages, Data Fetching and Parsing. The algorithm takes as input the Github Personal Access Token, *token*, the repository *repo* chosen to fetch the data and artifacts, and the endpoints whose connection with Github API was implemented. The outputs are the artifacts and relevant information that will be present in the SCIR, as well as a log file containing relevant information about the errors and warnings during the process.

Algorithm 1 SCIR Generation Pseudocode - Data Fetching and Parsing

Inputs:

Github Personal Access Token: *token*

Github Repository: *repo*

Github API Endpoints: *endpoints*

Outputs:

Artifacts: *artifacts*

Log File: *log_file*

Fetching and Parsing Github Data

GITHUBAUTHENTICATION(*token*)

data $\leftarrow \emptyset$

log_file $\leftarrow$ NEWFILE()

for endpoint in endpoints **do**

endpoint_data $\leftarrow$ FETCHDATA(endpoint)

if *endpoint_data*

data += *endpoint_data*

else if *endpoint_data* is not valid

log_file.add(EndpointError)

else if *endpoint_data* = $\emptyset$

log_file.add(EndpointWarning)

end if

end for

artifacts $\leftarrow \emptyset$

for information in data **do**

information $\leftarrow$ PARSEDATA(*information*)

artifact $\leftarrow$ FETCHARTIFACTS(*information*)

artifacts += *artifact*

end for

The pseudocode in Algorithm 2 displays the process of the last stage, Document Generation. The algorithm takes as input the artifacts and information produced in the first two stages, *artifacts*, and it outputs the SCIR report fully completed.

Algorithm 2 SCIR Generation Pseudocode - Document Generation

Inputs:
 Artifacts: *artifacts*

Outputs:
 SCIR Report: *scir_report*

Constructing the SCIR document
scir_report $\leftarrow$ NEWFILE()
for artifact in artifacts **do**
 if artifact **AND** artifact.stage
 ADDARTIFACTTOREPORTSTAGE(artifact)
 else if artifact is not valid
 scir_report.add(ArtifactError)
 else if artifact = $\emptyset$
 scir_report.add(ArtifactWarning)
 else
 ADDARTIFACTTOREPORTMISCELLANEOUS(artifact)
 end if
end for
return *scir_report*

6.3 Document Traceability Automation

After implementing the main executable script, there were still remaining DO-178C standard requirements whose automation could be automated. In order to automate this requirements verification, prior to the delivery of the certifiable documentation, one valuable executable script to be implemented would be a Document Traceability Script.

In order to meet the highest standards of safety and quality, the critical software certification process requires a set of documents that shall be delivered prior to this process. These documents contain different information about the software development activities conducted by the critical software development company. Due to the high cost and responsibility involved in a V&V and certification process, each delivery must be thoroughly inspected to confirm its faultlessness.

The purpose of the purposed executable script is to reduce to zero the possibility of delivery without the totality of the certifiable documentation. This objective is reached by iterating through the produced certifiable data package, looking for the files with mandatory presence while logging the missing files, and providing output to the developers. This script can also be used to standards other than the DO-178C standard, providing a good tool for the safety-critical software development industry.

Figure 6.3 represents the flowchart of the Document Traceability Tool.

To achieve this flowchart, there were three steps to be considered:

- Documentation fetching;
- Script execution;
- Logging;

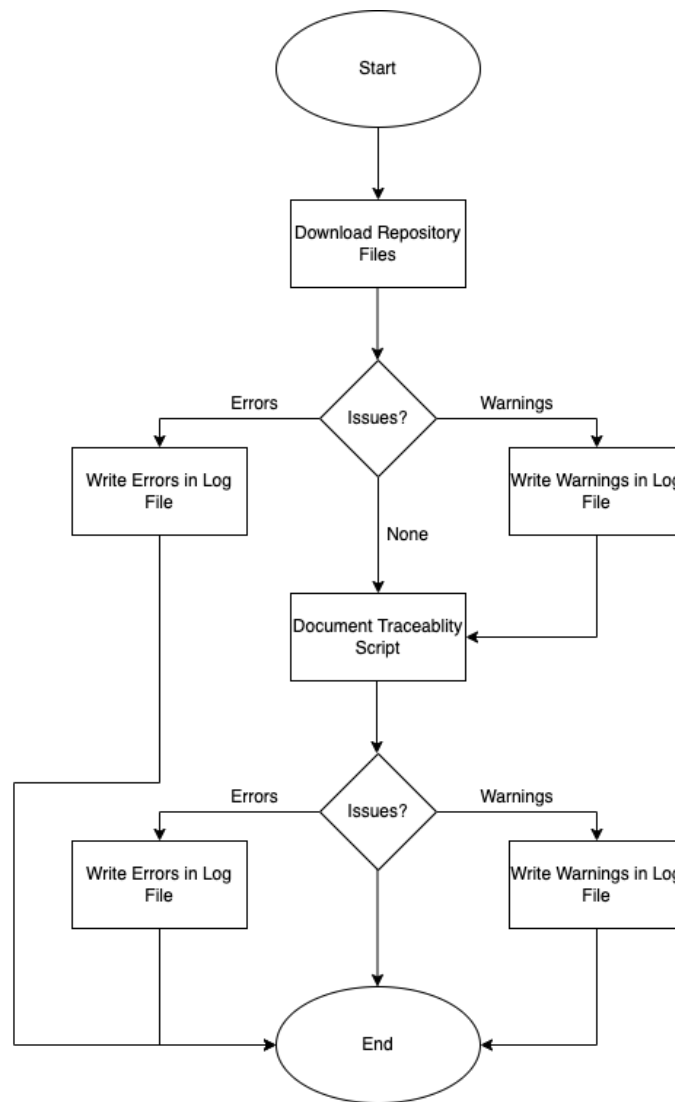


Figure 6.3: Document Traceability Automation Flowchart

The first of these steps is achieved by reusing the SCIR Generation endpoints and communication with the Github API by fetching all of the documents present in the critical system's software repository. By doing so, we retrieve both source code files and the software system's documentation. If any issue occurs during this phase, logging of the errors or warnings happens.

After this first step, there is the script execution. While the script executes, and similarly to the first step, there is the logging of warnings and errors into a log file. After the execution of the script, there are two possible scenarios: **1)** If the execution of the script had no inherent issues and the documents are all present in the data package, the log file is not generated, meaning that the certifiable documentation is complete and the script execution occurred without any issue or **2)** there were issues during the script execution, whether in the documentation fetching or the verification of the present document files, creating a log file containing information about the issues.

The pseudocode in Algorithm 3 shows the functioning of the Document Traceability Automation Tool.

Algorithm 3 Document Traceability Automation

Inputs:Github Personal Access Token: *token*Github Repository: *repo*Repository File Endpoint: *endpoint***Outputs:**Log File: *log_file***Document Traceability Script**GITHUBAUTHENTICATION(*token*)*data* $\leftarrow \emptyset$ *log_file* $\leftarrow$ NEWFILE()*data* $\leftarrow$ FETCHDATA(*repo*,*endpoint*)**if** *data* is not valid *log_file.add*(Error) **return** *log_file***else if** *artifact* = $\emptyset$ *log_file.add*(Warning)**else** [*warnings*, *errors*] $\leftarrow$ DOCUMENTTRACEABILITYSCRIPT(*data*)**end if****for** *warning* in *warnings* **do** *log_file.add*(*warning*)**end for****for** *error* in *errors* **do** *log_file.add*(*error*)**end for****return** *log_file*

The algorithm takes as input the Github Personal Access Token, *token*, that will allow the authentication with the Github API, the repository from where the data is fetched, *repo*, and the endpoint, *endpoint*, that will allow the connection with the Github API. The output is the log file, *log_file*, containing the detailed issues during the process of the execution of the script.

6.4 Known Limitations

Both of these scripts were designed and implemented within the scope of CSW's critical software development. The tech stack used for its implementation is an alternative to the one currently used by the company (Jira, Confluence, and BitBucket), to produce more generic executable scripts that can improve automation and agility in wider contexts, simply using Github. This was done in collaboration with the PhD research [33].

When focusing on the implementation itself, despite the broad features of Github and consequently its API, there are still some points that could be implemented that would enrich the already implemented scripts, namely:

- Issues with tags and files;
- Commits with labels;
- Specific files with labels/tags;
- Actions artifacts with labels/tags;
- More specific endpoints;

These features would allow, specifically in the SCIR Generation Script, a better specificity on the endpoints used and better artifact collection points (not only in pull requests and commits, as implemented in this stage) that could benefit the implemented tool.

Furthermore, some guidelines need to be followed for the flawless functioning of the SCIR Generation Tool. There needs to be a label associated to pull requests in order to fetch the required artifacts to fill in the SCIR. The fetched files also need to comply with some documented extensions, i.e., docx, pdf, zip, *et cetera*.

The fact of merging all of the specific aspects of a process overview, i.e., requirements, PBI, source code, etc., on a single version control system can also be a limitation since tools such as Jira and Confluence provide greater value due to their separated API from Github or BitBucket, meaning that data can be fetched with more precision depending on the required information.

6.5 Implemented Work and the DMT

As stated in Section 1.4, with a visual representation in Figure 1.1, this work proposed the implementation of a DMT, contributing to the improvement of agility and automation of documentation-related. As a consequence of the lack of time to implement such a complex tool, a fully implemented DMT was unreachable. The implemented executable scripts constitute small tools that can later be implemented in a DMT.

Figure 6.4 shows the current state of the DMT, with the currently implemented and possible future tools that would allow the completeness of this software system. Each script is depicted as a gear of a multi-phased and complex system, requiring each of the gears to work to have a fully automated DMT. The black gears are the implemented scripts, while the grey gears are the automation scripts that still can be implemented (with some examples envisioned during this dissertation).

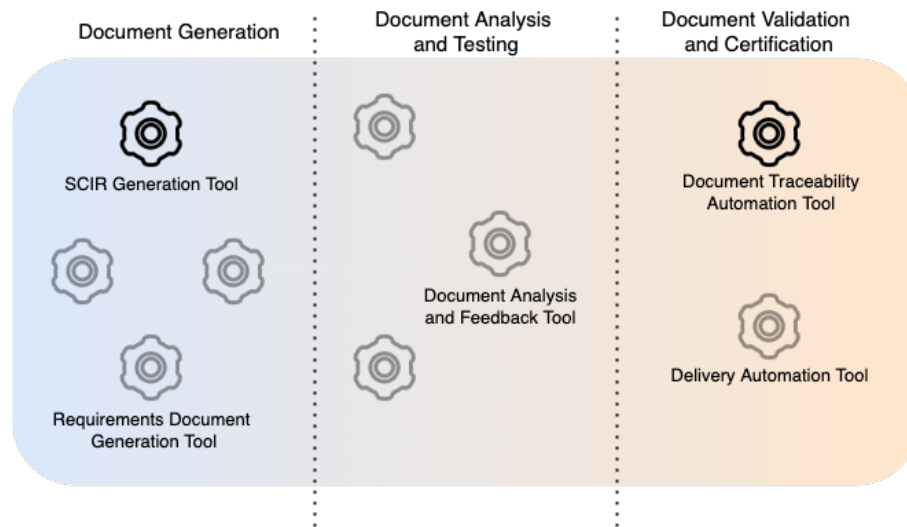


Figure 6.4: Current state of the DMT

6.6 Summary

This chapter describes the implementation of a SCIR Generation Tool and a Document Traceability Automation Tool which allow documentation automation within the critical software development industry.

In Section 6.1, the methodology that led to the implementation of the executable scripts is presented.

The SCIR Generation tool is presented in Section 6.2, outlining the details of the process overview being case studied as part of the PhD research in collaboration with CSW [33] and the implementation details, presenting the overview of the script, detailing the tech stack, architecture, and pseudocode.

Section 6.3 focuses on the Document Traceability Automation Tool and, similarly to the previous detailed Section, an overview of the script is detailed, containing its flowchart and implementation characteristics.

As for Section 6.4, we describe the limitations of the developed scripts, such as the restrictions imposed by the Github API.

Finally, in Section 6.5, there is the exhibition of the actual state of the proposed DMT with the implemented executable scripts and with possible scripts to be implemented in the future.

Chapter 7

Evaluation

This chapter outlines the experimental process used to evaluate our proposed solution. Section 7.1 presents the experimental setup used for the experiments. Section 7.2 describes the experiments conducted and the discussion of its results, while Section 7.3 includes the hypothesis evaluation and research question answers.

7.1 Experimental Setup

To evaluate the proposed solution, different tests were made on both implementation scripts.

The experiments and tests were performed in a Macbook Pro 2021, with an M1 8 core CPU and 16GB of RAM, running macOS Monterrey 12.0.1. The Python version used to implement the scripts was 3.10.2.

Given the confidentiality associated with CSW's projects, no access was granted to actual or *post-mortem* projects. To find an alternative to thorough testing, unit tests were implemented on top of different methods:

- Authentication with Github API;
- Github API Endpoints Data Fetching;
- Data Parsing;
- SCIR Generation;
- Data Download;
- Document Traceability Script;

These tests were conducted having as data repositories, since CSW's archives were confidential, we used Github private repositories created and configured for testing purposes. The account

used had access and privileges on the repository, and the respective Github Personal Access Token was configured with the needed permission for accessing the repository's data.

7.2 Results & Discussion

Given the type of the implemented work, executable scripts, its long-term utility is to be implemented in a version control system's pipeline of a critical software system. To validate our tools, there was the focus on assuring three main characteristics inherent to software and tests present in a pipeline:

- Correctness;
- Maintainability;
- Performance.

7.2.1 Correctness

The first characteristic inherent to the produced executable scripts is correctness. Correctness evaluates the software's accuracy, i.e., the quality of being free from error. To assess this vital aspect, there is the need to compare the expected results and the results produced from the script executions.

When concerning SCIR generation, given that the document will be reviewed and certified by formal authorities, there are rules to be followed. SCIR must follow a standard provided by CSW for this work, which allows the correctness evaluation of the tests' results. According to a SCIR standard, the template file is shown in Appendix B. To achieve a similar result, unit tests were made for each generation process stage. In case of a severe flaw, the script aborts. In case of defects during the process, there is problem logging to attain more information for the developers, despite the creation of the SCIR. Otherwise, when no warnings/errors are observed, there is a standard SCIR generation. One of the tests is depicted in Appendix C.

The Document Traceability Automation Tool was also the target of unit testing, separated by the three modules of the tool: document fetching and script execution and logging. This tool can be configured to have more control over which files it will search. Appendix D shows a log file of a test validating the presence of seven files with two of those missing.

These tests allow us to verify that both tools are fault-tolerant, producing the expected outputs and thus granting correctness.

7.2.2 Maintainability

One other characteristic that must be intrinsic to the implemented tools is maintainability. This characteristic refers to the easiness, precision, safety, and economy of changing the software system, regardless of the objective. Concretely, in the critical software systems development industry,

new requirements or a change in the standard used require ductility from both software and tools used.

Aiming towards this goal, both the SCIR Generation Tool and the Document Traceability Tool was implemented following a modular architecture, encapsulating each of the developed features and thus, making each task-independent. An example of this practice is the reusability of the Github API communication module in each of the scripts.

SCIR Generation Tool's architecture, as depicted in Figure 6.2 and above-mentioned, being modular allows this executable script's inclusion within a pipeline of a critical software system given the easiness of correcting errors/bugs, implementing new features or even changing the context of the standard followed to the development.

As for the Document Traceability Automation Tool, beyond the modular architecture, there is also the implementation of a structure that allows the utilization of other standards. Currently, only the DO-178C and EN50128 standards are applicable (depending on which files the tool will look for) however, maintainability is assured with the easiness of adding new standards and features.

7.2.3 Performance

Lastly, the performance of both of the tools has a great impact on whether their usage can be viable. To introduce both of these tools in a pipeline, the performance, mainly in terms of time spent on each script, needed to be measured, evaluated, and compared to the currently implemented techniques and methodologies used within CSW.

An interview process was conducted with CSW's professionals in order to compare the produced executable scripts and the currently applied methods in the company. In order to do so, the time spent for each approach was calculated/gathered since this is an important indicator, especially for a pipeline. The guide used for the interview process is displayed in Appendix E

Firstly, synthetic tests were made to the SCIR Generation Tool. Each of these executions was made with 5, 15, 45, and 100 files composing the SCIR. Figure 7.1 shows the performance of the executions.

The estimated period to manually build a SCIR, according to the interviews conducted with CSW professionals, varies between 8 and 40 hours for 500 1000 files, so we can conclude, despite the tests not covering this large number of files, that the SCIR Generation Tool outperforms the currently applied methods considering the different orders of magnitude, providing a reliable result.

The Document Traceability Tool was also the target of synthetic tests without data downloading to compare the performance of its execution as opposed to the estimated time spent by CSW's professionals. Similar to the SCIR Generation Tool's tests, 5, 15, 45, and 100 files were used for this purpose. These tests have the particularity of not including the download phase prior to the certification of the data package integrity. Figure 7.2 shows the performance comparison.

According to the interviews conducted with CSW's professionals, the V&V of the data package wholeness process takes between 1 hour and 4 hours, meaning that the usage of the Document

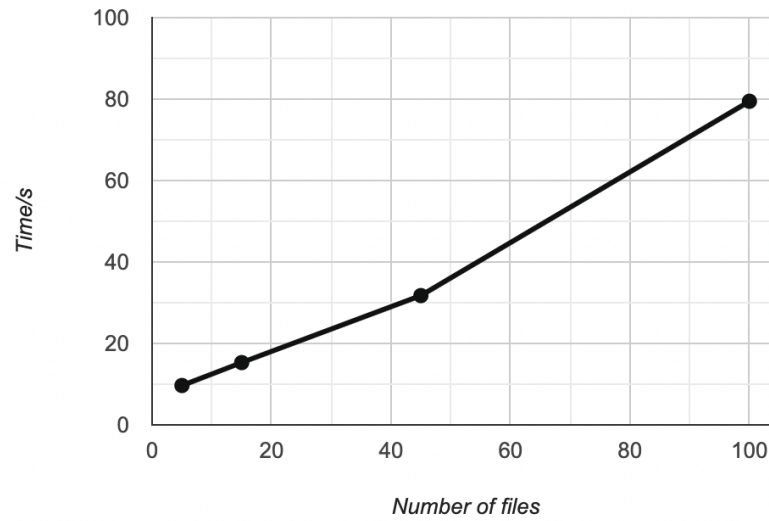


Figure 7.1: SCIR Generation Tool synthetic tests performance

Traceability Automation can benefit the performance of the software certification and release procedures.

7.3 Hypothesis and Research Questions Evaluation

The evaluation process was intended to provide answers to the research questions and the hypothesis defined in Section 4.4. Therefore, we re-visit them, summarizing our findings and providing a clear answer:

RQ2: Can the automation of the documentation-related processes benefit the software certification procedure?

The developed work during this dissertation proved that it is possible to automate document-related processes in a critical software development context, diverging from the current Waterfall methodologies applied. As shown in Figures 7.1 and 7.2 there are inherent benefits to the automation of document-related processes, especially in terms of developers' workload reduction and time-saving.

The software certification procedures conducted by the company prior to the delivery of the software and documentation to the regulatory entity are positively impacted by the usage of a DMT since it deals with document V&V and certification. Using the implemented Document Traceability Automation Tool, for instance, there is partial automation of these procedures, which contributes to a safer and more meticulous release of the certifiable data package.

RQ3: How can we optimize a DMT to make it *quasi-autonomous*?

The implementation of executable scripts and, consecutively, a DMT aims for the automation of the documentation-related processes, reducing the workload from software developers. In order to fully reach this goal, an autonomous tool would be necessary. Given the already implemented

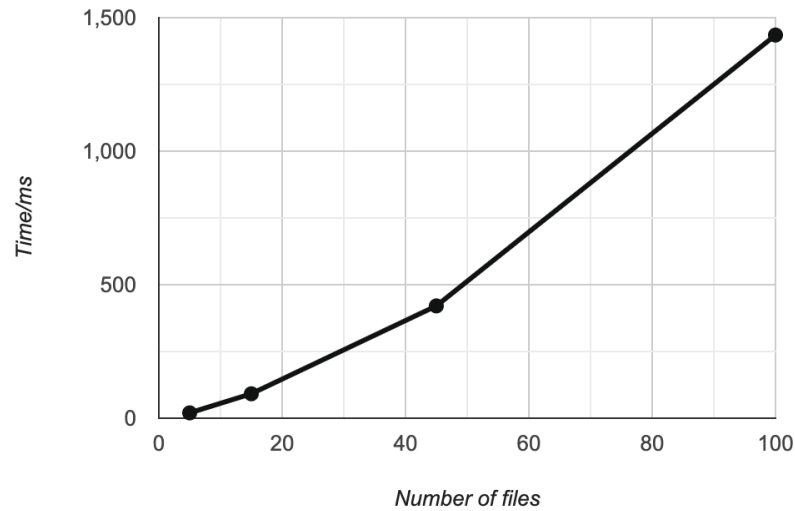


Figure 7.2: Document Traceability Automation Tool synthetic tests performance

tools, we can ensure that full autonomy is beyond reach. However, a *quasi-autonomous* state is possible.

Given the implemented work, there is always the need for human contribution for a faultless DMT, whether by analyzing the logs, containing information about issues during the procedures, maintaining the scripts, including them in the pipeline, or providing the needed data/information for the scripts functioning, i.e., repository, requirements, *et cetera*.

RQ4: Can a DMT output certifiable documentation independently?

With the production of the SCIR Generation Tool, we can prove that a DMT can produce certifiable documentation independently. Despite being just one of the composing executable scripts of the DMT, it leads the way into new possible executable scripts that can generate the different documents composing the certifiable data package.

The interviews conducted with CSW's professionals also mentioned, in light of the produced results, the need for the generation of more certifiable documents and release notes.

When included in a pipeline of a repository, a DMT can unassistedly produce documentation, analyze it, validate and certificate it.

RQ1 was not fully targeted since the focus of this dissertation work was the implementation of a DMT, making the documentation-related processes more agile, instead of studying Agile methodologies thoroughly.

Finally, after answering the pivotal research questions, we can evaluate the hypothesis claimed by this work:

"Document automation through a DMT within the context of critical software systems can improve the software development process, achieving greater efficiency, quality, and effectiveness of the document and certification-related processes."

Taking into consideration the answers to this dissertation's research questions, we can reach the conclusion that including a DMT responsible for document automation within the critical software projects development can help achieve greater efficiency, quality, and effectiveness of the document and certification-related processes by (1) relieving workload from the developers, automating the documentation-related processes and reducing human interaction to a bare minimum, (2) increasing the performance, specifically in terms of increasing accuracy and time reduction, of the document production and certification of document-related processes, and (3) there is the possibility to adapt a DMT or any of the independent executable scripts to any standard that needs to be followed in the critical software development industry.

7.4 Summary

In this chapter, the results drawn from the evaluation process are presented. Section 7.1 defines the system in which the experiments occurred, then Section 7.2 describes the testing objectives and its analysis, focusing on the inherent characteristics of executable scripts present in a software and documentation development pipeline (Subsections 7.2.1, 7.2.2, 7.2.3).

Section 7.3 re-visits the research questions and hypothesis that driven this dissertation, summing up the conclusions taken from this work.

Chapter 8

Conclusion

This chapter demonstrates an overview of this dissertation and contours future research and work. Section 8.1, the key conclusions and observations are presented, as well as the main contributions of this work. Finally, in Section 8.2, possible future features and research directions are presented.

8.1 Conclusions

Critical software systems impact millions of lives every day and depend on the reliability and defectless processes and activities. Thus, there is the need to improve the agility of these systems, targeting the automation of its development, testing, V&V, and certification.

Documentation has proven to be one of the obstacles to the implementation of Agile methodologies, as opposed to the Waterfall and plan-driven methods currently applied, and to better efficiency of the V&V and certification processes performed by regulatory bodies.

After the analysis of state of the art, providing an insight on the document-related processes automation barriers and the DO-178C standard, the examination and classification of this standard's procedure checklist in terms of automation types provided enlightenment on the tools that needed to be designed and implemented. Accordingly with CSW's professionals, the priority focused on the implementation of a SCIR Generation Tool and a Document Traceability Automation Tool.

The implementation of these tools focused on the principles of (1) automating its process, making it *quasi-autonomous* and, thus, reducing the professionals' workload, and (2) making the tools as adaptable as possible in order to reuse them in different contexts and standards on the critical software development industry. These tools compose the first checkpoint to the implementation of a complete DMT, whose objective is to deal with all of the documentation of a project, making it automatically verifiable and certifiable.

Considering the overall results and analysis of the experiments, we can conclude that a DMT implementation can provide a better experience to the critical software industry professionals, providing better accuracy, efficiency, and performance of document-related processes.

The work produced during this dissertation resulted in the following contributions:

DO-178C standard Development Processes Automation Checklist The characterization of the different processes of the DO-178C standard according to the type of automation that can be applied in order to enhance its efficiency.

SCIR Generation Tool An executable script capable of building a SCIR given a repository with the documents from the different stages of a company's process overview.

Document Traceability Automation Tool A tool responsible for the V&V of the integrity of the deliverable data package to critical software regulatory entities.

8.2 Future Work

The implemented tools contain limitations, described in Section 6.4 that can be addressed in the future. Additionally, there are also some aspects that can be analyzed in the future.

Firstly, and most importantly, given CSW's cooperation during this dissertation's work, there is the need to satisfy the needs of its project overview. In the interviews conducted with CSW professionals, there were some insights about what could possibly be the next checkpoints for the DMT:

- Different documents generation (i.e., software design, requirements, software coding process, *et cetera*;
- Release Notes automated generation;
- Traceability tools;
- Impact analysis script in graph databases;
- Configuration Management Check-Up Tool (validating functional and physical attributes and its cohesion with requirements, design and operational information).

Secondly, an additional thorough analysis of Appendix A is needed to perceive what are the next components of the DMT to be implemented, according to the importance of each of the tasks. Some scripts may include the automation of multiple points of the checklist, so their priority should be greater than the ones that don't. This will allow the conclusion of a fully functional DMT, the main goal of this dissertation's work.

Finally, the ideation, prototyping, and implementation of different executable scripts, tests, software, *et cetera* in order to conclude the DMT and integrate it into a pipeline of a critical software project, accomplishing the main objective of this dissertation's work.

References

- [1] Federal Aviation Administration. Federal Aviation Administration: Order 8110.49 - Software Approval Guidelines. Technical report, Federal Aviation Administration, Washington, D.C., October 2017.
- [2] Ademar Manuel Teixeira de Aguiar. *Framework documentation : A minimalist approach*. PhD thesis, Faculty of Engineering of the University of Porto, 2003. Accepted: 2012-03-12T11:19:07Z.
- [3] Mohammad Ahmadi Achachlouei, Omkar Patil, Tarun Joshi, and Vijayan N. Nair. Document Automation Architectures and Technologies: A Survey. Technical report, Corporate Model Risk, September 2021. Publication Title: arXiv e-prints ADS Bibcode: 2021arXiv210911603A Type: article.
- [4] Atlassian. Atlassian | Software Development and Collaboration Tools. Accessed in February 2022.
- [5] Atlassian. Jira | Issue & Project Tracking Software. Accessed in May 2022.
- [6] Nanduri Balajee. What is CI/CD Pipeline?, January 2020.
- [7] Youssef Bassil. A Simulation Model for the Waterfall Software Development Life Cycle. Technical Report arXiv:1205.6904, arXiv, May 2012. arXiv:1205.6904 [cs] type: article.
- [8] Kent Beck and Et al. Manifesto for Agile Software Development. <http://agilemanifesto.org/>.
- [9] Emmanuel Chenu. Agility and Lean for Avionics. *6th european congress on real time software and systems*, page 13, 2009.
- [10] Gabriele Comoretto, Leanne Guy, William O’Mullane, Keith Bechtol, Jeffrey L. Carlin, Brian van Klaveren, Austin Roberts, and Jonathan Sick. Documentation automation for the verification and validation of Rubin Observatory software. In George Z. Angeli and Philippe Dierickx, editors, *Modeling, Systems Engineering, and Project Management for Astronomy IX*, page 16, Online Only, United States, December 2020. SPIE.
- [11] John Cullyer. Safety critical systems. *Microprocessors and Microsystems*, 17(1):2, 1993.
- [12] Digital.ai. 14th Annual State Of Agile Report | Digital.ai.
- [13] Digital.ai. 15th Annual State Of Agile Report | Digital.ai.
- [14] Radio Technical Commission for Aeronautics. Software Considerations in Airborne Systems and Equipment Certification, 1992.

- [15] Andrew Forward and Timothy C. Lethbridge. The relevance of software documentation, tools and technologies: a survey. In *Proceedings of the 2002 ACM symposium on Document engineering*, DocEng '02, pages 26–33, New York, NY, USA, November 2002. Association for Computing Machinery.
- [16] Milad Hanna, Nahla El-Haggar, and Mostafa Sami. A Review of Scripting Techniques Used in Automated Software Testing. *International Journal of Advanced Computer Science and Applications*, 5(1), 2014.
- [17] Lise Tordrup Heeager and Peter Axel Nielsen. A conceptual model of agile software development in a safety-critical context: A systematic literature review. *Information and Software Technology*, 103:22–39, November 2018.
- [18] Vance Hilderman. Understanding DO-178C Software Certification: Benefits Versus Costs. In *2014 IEEE International Symposium on Software Reliability Engineering Workshops*, pages 114–114, November 2014.
- [19] Vance Hilderman and Tony Baghi. *Avionics Certification: A Complete Guide to DO-178 (software), DO-254 (hardware)*. Avionics Communications, 2007. Google-Books-ID: n9EKpCNIC_0C.
- [20] Mike Hinchey and Lorcan Coyle. Evolving Critical Systems: A Research Agenda for Computer-Based Systems. In *2010 17th IEEE International Conference and Workshops on Engineering of Computer Based Systems*, pages 430–435, Oxford, England, 2010. IEEE.
- [21] Gibrail Islam and Tim Storer. A case study of agile software development for safety-Critical systems projects. *Reliability Engineering & System Safety*, 200:106954, August 2020.
- [22] D. Janzen and H. Saiedian. Test-driven development concepts, taxonomy, and future direction. *Computer*, 38(9):43–50, September 2005. Conference Name: Computer.
- [23] Katja Karhu, Tiina Repo, Ossi Taipale, and Kari Smolander. Empirical Observations on Software Testing Automation. In *2009 International Conference on Software Testing Verification and Validation*, pages 201–209, April 2009. ISSN: 2159-4848.
- [24] Rashidah Kasauli, Eric Knauss, Benjamin Kanagwa, Agneta Nilsson, and Gul Calikli. Safety-Critical Systems and Agile Development: A Mapping Study. In *2018 44th Euromicro Conference on Software Engineering and Advanced Applications (SEAA)*, pages 470–477, August 2018.
- [25] J.C. Knight. Safety critical systems: challenges and directions. In *Proceedings of the 24th International Conference on Software Engineering. ICSE 2002*, pages 547–550, May 2002.
- [26] Robert Lankester. Implementing Document Automation: Benefits and Considerations for the Knowledge Professional. *Legal Information Management*, 18(2):93–97, June 2018. Publisher: Cambridge University Press.
- [27] Craig Larman. *Agile and Iterative Development: A Manager's Guide*. Addison-Wesley Professional, 2003.
- [28] Jae-Seung Lee, Seung-Eun Yang, Jong-Wook Choi, Yee-Jin Cheon, and Jeong-Oh Yun. Introduction to Automatic Generation of Design Documents for Flight Software using Doxygen. *Proceedings of the Korea Information Processing Society Conference*, pages 844–847, 2012. Publisher: Korea Information Processing Society.

- [29] Sun Linling, Zhang Wenjin, and Tim Kelly. Do safety cases have a role in aircraft certification? *Procedia Engineering*, 17:358–368, January 2011.
- [30] Thor Myklebust and Tor Stålhane. Agile Development. In Thor Myklebust and Tor Stålhane, editors, *The Agile Safety Case*, pages 19–46. Springer International Publishing, Cham, 2018.
- [31] Richard F. Paige, R., Ramon Charalambous, Xiaocheng Ge, and Phillip J. Brooke. Towards Agile Engineering of High-Integrity Systems. In Michael D. Harrison and Mark-Alexander Sujan, editors, *Computer Safety, Reliability, and Security*, pages 30–43, Berlin, Heidelberg, 2008. Springer Berlin Heidelberg.
- [32] Jan Peleska. Test automation for safety-critical systems: Industrial application and future developments. volume 1051, pages 39–59. January 2006.
- [33] J. Eduardo Ferreira Ribeiro. *Towards Continuous Certification of Software Systems for Aerospace*. PhD thesis, Faculty of Engineering of the University of Porto, 2022. (*Under development*).
- [34] Darrell K Rigby, Jeff Sutherland, and Andy Noble. Agile at scale: How to go from a few teams to hundreds. *Harvard Business Review*, page 10, 2018.
- [35] Jussi Ronkainen and Pekka Abrahamsson. Software Development Under Stringent Hardware Constraints: Do Agile Methods Have a Chance? *arXiv:1711.08637 [cs]*, 2675:73–79, 2003. arXiv: 1711.08637.
- [36] Inc. RTCA. Software Considerations in Airborne Systems and Equipment Certification. *Radio Technical Commission for Aeronautics, Document*, page 144, 2011.
- [37] Benjamin Shestakofsky. Working Algorithms: Software Automation and the Future of Work. *Work and Occupations*, 44(4):376–423, November 2017. Publisher: SAGE Publications Inc.
- [38] Ahmed Sidky and James Arthur. Determining the Applicability of Agile Practices to Mission and Life-critical Systems. *arXiv:cs/0701010*, January 2007. arXiv: cs/0701010.
- [39] Critical Software. Critical Software. <https://criticalsoftware.com/>. Accessed in February 2022.
- [40] Ian Sommerville. Critical systems, October 2014.
- [41] Write the Docs. DocOps — Write the Docs. Accessed in February 2022.
- [42] Yan Ting Wong Tiky. Software Development Life Cycle. page 43.
- [43] Lise Tordrup and Peter Nielsen. Meshing agile and plan-driven development in safety-critical software: a case study. *Empirical Software Engineering*, 25:1–28, March 2020.
- [44] Steven H. VanderLeest and Andrew Buter. Escape the waterfall: Agile for aerospace. In *2009 IEEE/AIAA 28th Digital Avionics Systems Conference*, pages 6.D.3–1–6.D.3–16, October 2009. ISSN: 2155-7209.
- [45] Matti Vuori. Agile Development of Safety-Critical Software. Technical report, Tampere University of Technology, 2011.

- [46] Laurie Williams. Agile Software Development Methodologies and Practices. In Marvin V. Zelkowitz, editor, *Advances in Computers*, volume 80 of *Advances in Computers*, pages 1–44. Elsevier, January 2010.
- [47] Andrew Wils, Stefan Van Baelen, Tom Holvoet, and Karel De Vlamincx. Agility in the Avionics Software World. In Pekka Abrahamsson, Michele Marchesi, and Giancarlo Succi, editors, *Extreme Programming and Agile Processes in Software Engineering*, Lecture Notes in Computer Science, pages 123–132, Berlin, Heidelberg, 2006. Springer.
- [48] Junbeom Yoo, Eunkyong Jee, and Sungdeok Cha. Formal Modeling and Verification of Safety-Critical Software. *IEEE Software*, 26(3):42–49, May 2009. Conference Name: IEEE Software.
- [49] Wonkeun Youn, Seung Hong, Kyung Oh, and Oh Ahn. Software Certification of Safety-Critical Avionic Systems: DO-178C and Its Impacts. *Aerospace and Electronic Systems Magazine, IEEE*, 30:4–13, April 2015.
- [50] Marvin Zelkowitz. *Advances in Computers: Advances in Software Engineering*. Elsevier, July 2004.

Appendix A

DO-178C Standard Classified Checklist

The following tables reflect the classification of the DO-178C standard checklist.

For each question respond with: - N/A: question not applicable to this function - Y: positive answer - N: negative answer For N/A answers a rationale shall be provided.		Automation Type
	Software Planning Process	
1	Plan for Software Aspects of Certification (baseline and release)?	Scripting and Manual
2	Software Development Plan (baseline and release)?	Scripting and Manual
3	Software Development Standards Document (baseline and release)?	Scripting and Manual
4	The software plans and standards have been approved, put under configuration control, and agreed with the certification authority ?	Manual and Pipeline/Tests
5	Planning Process Review (PPR) has been completed (meeting minutes and PPR were produced and under configuration control)?	Scripting and Pipeline/Tests
6	Software Requirements Data (baseline and release)?	Scripting and Manual
7	Software Requirements Data has been approved and put under configuration control?	Manual and Pipeline/Tests
8	System requirements, safety-related requirements and software requirements are all traceable?	Scripting and Pipeline/Tests
9	Software Requirements Process Review (SRPR) has been completed (meeting minutes and SRPR were produced and under configuration control)?	Scripting and Pipeline/Tests
10	Software Verification Results have been produced, analysed and put under configuration control, and the open problems were captured in Problem Reports?	Scripting, Manual and Pipeline/Tests
11	SRR checklist completed and under configuration control	Scripting, Pipeline/Tests
12	Software Design Description (baseline and release)?	Scripting and Manual
13	Software Reviews (SDR) has been completed (meeting minutes and SDPR were produced and under configuration control)?	Scripting and Manual
14	SDR checklist has been completed and configured?	Scripting and Manual
15	Source Code has been approved and put under configuration control?	Manual and Pipeline/Tests
16	Source Code files, used to produce the Executable Object Code and Parameter Data Item files is baseline and released?	Scripting and Manual
17	Source Code conforms to the standards?	Manual and Pipeline/Tests
18	Software Build Log File for compilation/linking of the source code units (baseline and release)?	Scripting and Manual
19	Software Coding Process Review (SCPR) has been completed (meeting minutes and SCPR were produced and under configuration control)?	Scripting and Pipeline/Tests
20	SQA Records have been produced, analysed and put under configuration control?	Scripting, Manual and Pipeline/Tests
21	Software Environment Life Cycle Configuration Index (baseline and release)?	Scripting and Manual
22	Software Configuration Index (baseline and release)?	Scripting and Manual
23	Software Analysis Report with the resulting logs from the compilation, linking and loading of the Executable Object Code and Parameter Data Item files into the target computer (baseline and release)?	Scripting and Manual
24	Software Executable Object Code and Parameter Data Item files have been produced, integrated with the target hardware and under configuration control?	Manual and Pipeline/Tests
25	Software build and link log files has been produced checked for error/warning free compilation/linking of the source code units, and under configuration control?	Pipeline/Tests

26	Software Integration Process Review (SIPR) has been completed (meeting minutes and SIPR were produced and under configuration control)?		Scripting and Pipeline/Tests
27	The Executable Object Code and Parameter Data Item files can be regenerated from the archived Source Code?		Pipeline/Tests
28	Plans and standards are baselined and released?		Scripting and Manual
29	High-level requirements (including PDI requirements) are baselined and released?		Scripting and Manual
30	Low-level requirements and software architecture are baselined and released?		Scripting and Manual
31	Software Verification Cases and Procedures (baseline and release)		Scripting and Manual
32	Software Verification Results (includes review and analysis reports) (baseline and release)		Scripting and Manual
33	Full bidirectional traceability between system requirements and high-level requirements are baselined and released?		Scripting and Manual
34	Full bidirectional traceability between high-level requirements and low-level requirements are baselined and released?		Scripting and Manual
35	Full bidirectional traceability between low-level requirements and source code is baselined and released?		Scripting and Manual
36	Full bidirectional traceability between high/low-level requirements and high-level requirements-based software verification cases is baselined and released?		Scripting and Manual
37	Full bidirectional traceability between high/low-level requirements-based software verification cases and corresponding software verification procedures are baselined and released?		Scripting and Manual
38	Full bidirectional traceability between high/low-level requirements-based software verification procedures and corresponding software verification results are baselined and released?		Scripting and Manual
39	All high/low-level requirements-based software verification test results are successful (pass or deviations justified)?		Pipeline/Tests
40	Integration activities results (part of the Software Verification Results) are baselined and released (Executable Object Code and Parameter Data Items are systematically and successfully generated)?		Scripting and Manual
41	Source Code is baselined and released?		Scripting, Manual and Pipeline/Tests
42	Are all the test cases, test procedures, test results and coverage analysis results reviewed?		Pipeline/Tests
43	Software Configuration Management Records are available?		Scripting
44	Configuration items identification is completed and SCI and/or SECI are updated?		Manual
45	Work products were baseline before each review activity?		Manual
46	Work products were baseline after each review activity?		Manual
47	Were all change request procedures completed, with notification provided?		Manual, Pipeline/Tests
48	Were all problem reports completed and with notification provided with the evidence that they have been evaluated and their status recorded?		Manual, Pipeline/Tests
49	The approved software can be loaded successfully through the use of release instructions?		Manual, Pipeline/Tests
50	Quality Assurance Reports (including corrective actions and audit reports) is baselined and released?		Scripting and Manual
51	Software Accomplishment Summary (baselined and released)?		Scripting and Manual
52	Software Configuration Index is baselined and released?		Scripting and Manual
53	Plan for Software Aspects of Certification (baselined and released)?		Scripting and Manual
54	Software development processes and integral processes comply with approved software plans and standards?		Manual and Pipeline/Tests

55	The transition criteria for the software life cycle processes are satisfied?		Manual and Pipeline/Tests
1	Are all plans (i.e. PMP, SDP - containing the SCMP and the SQAP, SVP, PSAC, TQP) and standards up to date to accommodate any changes and are they being followed?		Manual
2	Has the software schedule and resource plan been updated to incorporate any changes?		Manual and Pipeline/Tests
3	Are data items and documentation at an acceptable state, and in accordance with the project processes, project schedule, and the requirements of this review?		Manual and Pipeline/Tests
4	Have all test cases and procedures been completed, reviewed, and placed under configuration in the developmental configuration management system.		Manual
5	Has completeness of traceability between software High-Level Requirements and software Low-Level Requirements been verified? <i>Note: some design review activities are N/A for Level C</i>		Manual
6	Are all of the software High-Level Requirement reviews complete with all of the artifacts under change control?		Manual
7	Are all of the design reviews (software architecture and software Low-Level Requirements) complete with all of the artifacts under change control? <i>Note: some design review activities are N/A for Level C</i>		Manual
8	Is the hardware unit under test production hardware, or demonstrated to be equivalent to production hardware in all ways that affect the testing that will be performed?		Manual
9	Is the hardware unit under test sufficiently controlled for formal test execution?		Manual
10	Is the hardware unit under test the right configuration for the software that will be tested?		Manual
11	Will the test environment be available in the timeframe needed?		Manual
12	Is all the test equipment software clearly and uniquely identified and under change control?		Manual
13	The candidate software is controlled as required by the project SCMP section of the SDP?		Manual and Pipeline/Tests
14	For any testing that will not be performed on the candidate software, the differences do not affect the testing that will be performed		Manual and Pipeline/Tests
15	Is the source to object code analysis complete, reviewed and under change control? <i>Note: some analysis activities are N/A for Level C</i>		Pipeline/Tests
16	Has a dry run of all tests been performed?		Pipeline/Tests
17	Are all tests debugged?		Pipeline/Tests
18	Have preliminary verification (test, analysis) results been analysed and reconciled?		Pipeline/Tests
19	The test procedures for the specified formal test have been approved, reviewed and are under change control (issued or controlled pending issue)?		Pipeline/Tests
20	Is the traceability of tests to HLRs and LLRs (if applicable) complete?		Manual
21	Have low-level tests (if applicable) been written, reviewed and approved?		Pipeline/Tests
22	Is the High-Level Requirements coverage and Low-Level Requirements coverage 100% complete or are the deficiencies justified?		Pipeline/Tests
23	Is the code structural coverage 100% complete or are the deficiencies justified?		Pipeline/Tests
24	Is the data and control coupling coverage 100% complete or are the deficiencies justified?		Pipeline/Tests
25	Where code instrumentation is used, such as to measure coverage, are tests re-executed without the instrumentation to ensure the instrumentation does not affect functionality?		Pipeline/Tests

26	Are all the code reviews complete with all of the artefacts under change control?		Pipeline/Tests
27	Have all process issues affecting test activities been resolved?		Manual and Pipeline/Tests
28	Test schedule has been established and is acceptable to all involved?		Manual
29	All open or deferred Change Requests/Problem Reports against the candidate software have been reviewed and dispositioned?		Manual
30	Are plans in place for handling problems with tests found during or after test execution? <i>Note: some LLR testing activities are NA for Level C</i>		Manual
31	Is the current measured resource usage acceptable?		Manual
32	Have tools been qualified as planned?		Manual
33	Are all tools under change control as per the applicable SCMP section of the SDP?		Scripting and Pipeline/Tests
34	Was the PDI verification procedure completed and verification report produced?		Manual and Pipeline/Tests
35	Are the test environment(s) conformed as specified in the software verification plan, software test plans, and/or software test procedures?		Manual and Pipeline/Tests
36	Is the test environment controlled?		Manual and Pipeline/Tests
37	Is the test environment capable of performing the tests?		Pipeline/Tests
38	Will the test environment produce repeatable results?		Manual
39	Is all the test equipment hardware clearly identified with part numbers, serial numbers and properly documented?		Scripting
40	Specialized software load procedures are documented, reviewed, and under change control (if any)?		Manual and Pipeline/Tests
41	Are artifacts from Technical Data Reviews identified and under change control?		Manual
42	Do software team lead and SQA agree formal test execution can proceed?		Manual

		Chapter	Requirements		Automation Type
SW Verification Process	1	11.14 Software Verification Results	The Software Verification Results are produced by the software verification process activities. Software Verification Results should: a. For each review, analysis, and test, indicate each procedure that passed or failed during the activities and the final pass/fail results.		Manual and Pipeline/Tests
SW Verification Process	2	11.14 Software Verification Results	The Software Verification Results are produced by the software verification process activities. Software Verification Results should: b. Identify the configuration item or software version reviewed, analyzed, or tested.		Manual and Pipeline/Tests
SW Verification Process	3	11.14 Software Verification Results	The Software Verification Results are produced by the software verification process activities. Software Verification Results should: c. Include the results of tests, reviews, and analyses, including coverage analyses and traceability analyses.		Manual and Pipeline/Tests
		Chapter	Requirements		
SW Verification Results	1	7.2.1 Configuration Identification	Activities include: a. Configuration identification should be established for the software life cycle data.		Manual
SW Verification Results	2	7.2.1 Configuration Identification	Activities include: b. Configuration identification should be established for each configuration item, for each separately controlled component of a configuration item, and for combinations of configuration items that comprise a software product.		Manual
SW Verification Results	3	7.2.1 Configuration Identification	Activities include: c. Configuration items should be configuration identified prior to the implementation of change control and traceability analysis.		Manual
SW Verification Results	4	7.2.1 Configuration Identification	Activities include: d. A configuration item should be configuration identified before that item is used by other software life cycle processes, referenced by other software life cycle data, or used for software manufacture or software loading.		Manual
SW Verification Results	5	7.2.1 Configuration Identification	Activities include: e. If the software product identification cannot be determined by physical examination (for example, part number plate examination), then the Executable Object Code and Parameter Data Item Files, if any, should contain configuration identification that can be accessed by other parts of the system or equipment. This may be applicable to field-loadable software.		Manual
SW Verification Results	6	7.2.2 Baselines and Traceability	Activities include: a. Baselines should be established for configuration items used for certification credit. Intermediate baselines may be established to aid in controlling software life cycle process activities.		Manual
SW Verification Results	7	7.2.2 Baselines and Traceability	Activities include: b. A software product baseline should be established for the software product and defined in the Software Configuration Index (see 11.16).		Scripting and Manual
SW Verification Results	8	7.2.2 Baselines and Traceability	Activities include: c. Baselines should be established in controlled software libraries, whether physical, electronic, or other, to ensure their integrity. Once a baseline is established, it should be protected from change. User-modifiable software is not included in the software product baseline, except for its associated protection and boundary components. Therefore, modifications may be made to user-modifiable software without affecting the configuration identification of the software product baseline.		Software
SW Verification Results	9	7.2.2 Baselines and Traceability	Activities include: d. Change control activities should be followed to develop a derivative baseline from an established baseline.		Software
SW Verification Results	10	7.2.2 Baselines and Traceability	Activities include: e. A baseline should be traceable to the baseline from which it was derived, if certification credit is sought for software life cycle process activities or data associated with the development of the previous baseline.		Pipeline/Test
SW Verification Results	11	7.2.2 Baselines and Traceability	Activities include: f. A configuration item should be traceable to the configuration item from which it was derived, if certification credit is sought for software life cycle process activities or data associated with the development of the previous configuration item.		Manual and Pipeline/Test
SW Verification Results	12	7.2.2 Baselines and Traceability	Activities include: g. A baseline or configuration item should be traceable either to the output it identifies or to the process with which it is associated.		Pipeline/Test

SW Verification Results	13	7.2.3 Problem Reporting, Tracking, and Corrective Action	Activities include: a. A Problem Report should be prepared that describes the process non-compliance with plans, output deficiency, or software anomalous behavior, and the corrective action taken, as defined in section 11.17.		Manual and Pipeline/Test
SW Verification Results	14	7.2.3 Problem Reporting, Tracking, and Corrective Action	Activities include: b. Problem reporting should provide for configuration identification of affected configuration item(s) or definition of affected process activities, status reporting of Problem Reports, and approval and closure of Problem Reports.		Scripting and Software
SW Verification Results	15	7.2.3 Problem Reporting, Tracking, and Corrective Action	Activities include: c. Problem Reports that require corrective action of the software product or outputs of software life cycle processes should invoke the change control activity.		Manual and Software
SW Verification Results	16	7.2.4 Change Control	Activities include: a. Change control should preserve the integrity of the configuration items and baselines by providing protection against their change.		Manual
SW Verification Results	17	7.2.4 Change Control	Activities include: b. Change control should ensure that any change to a configuration item requires a change to its configuration identification.		Manual
SW Verification Results	18	7.2.4 Change Control	Activities include: c. Changes to baselines and to configuration items under change control to produce derivative baselines should be recorded, approved, and tracked. Problem reporting is related to change control, since resolution of a reported problem may result in changes to configuration items or baselines.		Pipeline/Test
SW Verification Results	19	7.2.4 Change Control	Activities include: d. Software changes should be traced to their origin and the software life cycle processes repeated from the point at which the change affects their outputs. For example, an error discovered at hardware/software integration, that is shown to result from an incorrect design, should result in design correction, code correction, and repetition of the associated integral process activities. : It is generally recognized that early implementation of change control assists the control and management of software life cycle process activities.		Software and Pipeline/Tests
SW Verification Results	20	7.2.4 Change Control	Activities include: e. Throughout the change activity, software life cycle data affected by the change should be updated and records should be maintained for the change control activity.		Scripting
SW Verification Results	21	7.2.5 Change Review	Activities include: a. Assessment of the impact of the problem or proposed change on system requirements. Feedback should be provided to the system processes, including the system safety assessment process, and any responses from the system processes should be assessed.		Manual
SW Verification Results	22	7.2.5 Change Review	Activities include: b. Assessment of the impact of the problem or proposed change on software life cycle data identifying changes to be made and actions to be taken.		Manual
SW Verification Results	23	7.2.5 Change Review	Activities include: c. Confirmation that affected configuration items are configuration identified.		Manual
SW Verification Results	24	7.2.5 Change Review	Activities include: d. Feedback of Problem Report or change impact and decisions to affected processes.		Scripting and Manual
SW Verification Results	25	7.2.6 Configuration Status Accounting	Activities include: a. Reporting on configuration item identification, baseline identification, Problem Report status, change history, and release status.		Scripting and Manual
SW Verification Results	26	7.2.6 Configuration Status Accounting	Activities include: b. Definition of the data to be maintained and the means of recording and reporting status of this data.		Manual
SW Verification Results	27	7.2.7 Archive, Retrieval, and Release	Activities include: a. Software life cycle data associated with the software product should be retrievable from the approved source (for example, an archive at the developing organization or company).		Manual and Pipeline/Test
SW Verification Results	28	7.2.7 Archive, Retrieval, and Release	Activities include: b. Procedures should be established to ensure the integrity of the stored data, regardless of medium of storage, by: 1. Ensuring that no unauthorized changes can be made.		Scripting and Manual
SW Verification Results	29	7.2.7 Archive, Retrieval, and Release	Activities include: b. Procedures should be established to ensure the integrity of the stored data, regardless of medium of storage, by: 2. Selecting storage media that minimize regeneration errors or deterioration.		Manual

SW Verification Results	30	7.2.7 Archive, Retrieval, and Release	Activities include: b. Procedures should be established to ensure the integrity of the stored data, regardless of medium of storage, by: 3. Preventing loss or corruption of data over time. Depending on the storage media used, this may include periodically exercising the media or refreshing the archived data.		Manual
SW Verification Results	31	7.2.7 Archive, Retrieval, and Release	Activities include: b. Procedures should be established to ensure the integrity of the stored data, regardless of medium of storage, by: 4. Storing duplicate copies in physically separate archives that minimize the risk of loss in the event of a disaster.		Scripting and Manual
SW Verification Results	32	7.2.7 Archive, Retrieval, and Release	Activities include: c. The duplication process should be verified to produce accurate copies, and procedures should exist that ensure error-free copying of the Executable Object Code and Parameter Data Item Files, if any		Scripting and Manual
SW Verification Results	33	7.2.7 Archive, Retrieval, and Release	Activities include: d. Configuration items should be identified and released prior to use for software manufacture and the authority for their release should be established. As a minimum, the components of the software product loaded into the airborne system or equipment should be released. This includes the Executable Object Code and Parameter Data Item Files, if any, and may also include associated media for software loading.		Manual
SW Verification Results	34	7.2.7 Archive, Retrieval, and Release	Activities include: e. Data retention procedures should be established to satisfy airworthiness requirements and enable software modifications.		Software and Pipeline/Tests
SW Verification Results	35	7.4 Software Load Control	Whichever method is used, software load control should include: a. Procedures for part numbering and media identification that identify software configurations that are intended to be approved for loading into the airborne system or equipment.		Manual
SW Verification Results	36	7.4 Software Load Control	Whichever method is used, software load control should include: b. Whether the software is delivered as an end item or is delivered installed in the airborne system or equipment, records should be kept that confirm software compatibility with the airborne system or equipment hardware.		Scripting and Manual
SW Verification Results	37	7.5 Software Life Cycle Environment Control	The software life cycle environment tools are defined by the software planning process and identified in the Software Life Cycle Environment Configuration Index (see 11.15). Activities include: a. Configuration identification should be established for the Executable Object Code, or equivalent, of the tools used to develop, control, build, verify, and load the software.		Manual and Software
SW Verification Results	38	7.5 Software Life Cycle Environment Control	The software life cycle environment tools are defined by the software planning process and identified in the Software Life Cycle Environment Configuration Index (see 11.15). Activities include: b. The SCM process for controlling qualified tools should comply with the objectives associated with Control Category 1 or Control Category 2 data (see 7.3), according to the guidance provided by section 12.2.3.		Manual
SW Verification Results	39	7.5 Software Life Cycle Environment Control	The software life cycle environment tools are defined by the software planning process and identified in the Software Life Cycle Environment Configuration Index (see 11.15). Activities include: c. Unless section 7.5b applies, the SCM process for controlling the Executable Object Code, or equivalent, for tools used to build and load the software (for example, compilers, assemblers, and linkage editors) should comply with the objectives associated with Control Category 2 data, as a minimum.		Manual
		Chapter	Requirements		
SW Verification Cases and Procedures	1	11.13 Software Verification Cases and Procedures	Software Verification Cases and Procedures detail how the software verification process activities are implemented. This data should include descriptions of the: a. Review and analysis procedures: The scope and depth of the review or analysis methods to be used, in addition to the description in the Software Verification Plan.		Manual
SW Verification Cases and Procedures	2	11.13 Software Verification Cases and Procedures	Software Verification Cases and Procedures detail how the software verification process activities are implemented. This data should include descriptions of the: b. Test cases: The purpose of each test case, set of inputs, conditions, expected results to achieve the required coverage criteria, and the pass/fail criteria.		Pipeline/Tests
SW Verification Cases and Procedures	3	11.13 Software Verification Cases and Procedures	Software Verification Cases and Procedures detail how the software verification process activities are implemented. This data should include descriptions of the: c. Test procedures: The step-by-step instructions for how each test case is to be set up and executed, how the test results are evaluated, and the test environment to be used.		Manual and Pipeline/Tests
		Chapter	Requirements		

SW Verification Plan	1	11.3 Software Verification Plan	This plan should include: Organization. Organizational responsibilities within the software verification process and interfaces with the other software life cycle processes.		Manual
SW Verification Plan	2	11.3 Software Verification Plan	This plan should include: Independence. A description of the methods for establishing verification independence, when required.		Manual
	3	11.3 Software Verification Plan	This plan should include: Verification methods: A description of the verification methods to be used for each activity of the software verification process: 1. Review methods, including checklists or other aids.		Manual (contains the plan for other automation types)
SW Verification Plan	4	11.3 Software Verification Plan	This plan should include: Verification methods: A description of the verification methods to be used for each activity of the software verification process: 2. Analysis methods, including traceability and coverage analysis.		Manual (contains the plan for other automation types)
SW Verification Plan	5	11.3 Software Verification Plan	This plan should include: Verification methods: A description of the verification methods to be used for each activity of the software verification process: 3. Testing methods, including the method for selecting test cases, the test procedures to be used, and the test data to be produced.		Manual (contains the plan for other automation types)
SW Verification Plan	6	11.3 Software Verification Plan	This plan should include: Verification environment. A description of the equipment for testing, the testing and analysis tools, and how to apply these tools and hardware test equipment. Section 4.4.3b provides guidance on indicating target computer and simulator or emulator differences.		Manual (contains the plan for other automation types)
SW Verification Plan	7	11.3 Software Verification Plan	This plan should include: Transition criteria: The transition criteria for entering the software verification process.		Manual (contains the plan for other automation types)
SW Verification Plan	8	11.3 Software Verification Plan	This plan should include: Partitioning considerations: If partitioning is used, the methods used to verify the integrity of the partitioning.		Manual (contains the plan for other automation types)
	9	11.3 Software Verification Plan	This plan should include: Compiler assumptions: A description of the assumptions made by the applicant about the correctness of the compiler, linkage editor, or loader (see 4.4.2).		Manual (contains the plan for other automation types)
SW Verification Plan	10	11.3 Software Verification Plan	This plan should include: Reverification method: For software modification, a description of the methods for identifying, analyzing, and verifying the affected areas of the software and the changed parts of the Executable Object Code.		Manual (contains the plan for other automation types)
SW Verification Plan	11	11.3 Software Verification Plan	This plan should include: Previously developed software: For previously developed software, if the initial compliance baseline for the verification process does not comply with this document, a description of the methods to satisfy the objectives of this document.		Manual (contains the plan for other automation types)
SW Verification Plan	12	11.3 Software Verification Plan	This plan should include: Multiple-version dissimilar software: If multiple-version dissimilar software is used, a description of the software verification process activities (see 12.3.2).		Manual (contains the plan for other automation types)

		Chapter	Requirements	Automation Type
Software Configuration Management Process	1	7.2.1 Configuration Identification	a. Configuration identification should be established for the software life cycle data. Activities include:	Manual
Software Configuration Management Process	2	7.2.1 Configuration Identification	b. Configuration identification should be established for each configuration item, for each separately controlled component of a configuration item, and for combinations of configuration items that comprise a software product. Activities include:	Manual
Software Configuration Management Process	3	7.2.1 Configuration Identification	c. Configuration items should be configuration identified prior to the implementation of change control and traceability analysis. Activities include:	Manual
Software Configuration Management Process	4	7.2.1 Configuration Identification	d. A configuration item should be configuration identified before that item is used by other software life cycle processes, referenced by other software life cycle data, or used for software manufacture or software loading. Activities include:	Manual
Software Configuration Management Process	5	7.2.1 Configuration Identification	e. If the software product identification cannot be determined by physical examination (for example, part number plate examination), then the Executable Object Code and Parameter Data Item Files, if any, should contain configuration identification that can be accessed by other parts of the system or equipment. This may be applicable to field-loadable software. Activities include:	Manual
Software Configuration Management Process	6	7.2.2 Baselines and Traceability	a. Baselines should be established for configuration items used for certification credit. Intermediate baselines may be established to aid in controlling software life cycle process activities. Activities include:	Manual
Software Configuration Management Process	7	7.2.2 Baselines and Traceability	b. A software product baseline should be established for the software product and defined in the Software Configuration Index (see 11.16). Activities include:	Scripting and Manual
Software Configuration Management Process	8	7.2.2 Baselines and Traceability	c. Baselines should be established in controlled software libraries, whether physical, electronic, or other, to ensure their integrity. Once a baseline is established, it should be protected from change. : User-modifiable software is not included in the software product baseline, except for its associated protection and boundary components. Therefore, modifications may be made to user-modifiable software without affecting the configuration identification of the software product baseline. Activities include:	Software
Software Configuration Management Process	9	7.2.2 Baselines and Traceability	d. Change control activities should be followed to develop a derivative baseline from an established baseline. Activities include:	Software
Software Configuration Management Process	10	7.2.2 Baselines and Traceability	e. A baseline should be traceable to the baseline from which it was derived, if certification credit is sought for software life cycle process activities or data associated with the development of the previous baseline. Activities include:	Pipeline/Test
Software Configuration Management Process	11	7.2.2 Baselines and Traceability	f. A configuration item should be traceable to the configuration item from which it was derived, if certification credit is sought for software life cycle process activities or data associated with the development of the previous configuration item. Activities include:	Manual and Pipeline/Test
Software Configuration Management Process	12	7.2.2 Baselines and Traceability	g. A baseline or configuration item should be traceable either to the output it identifies or to the process with which it is associated. Activities include:	Pipeline/Test
Software Configuration Management Process	13	7.2.3 Problem Reporting, Tracking, and Corrective Action	a. A Problem Report should be prepared that describes the process non-compliance with plans, output deficiency, or software anomalous behavior, and the corrective action taken, as defined in section 11.17. Activities include:	Manual and Pipeline/Test
Software Configuration Management Process	14	7.2.3 Problem Reporting, Tracking, and Corrective Action	b. Problem reporting should provide for configuration identification of affected configuration item(s) or definition of affected process activities, status reporting of Problem Reports, and approval and closure of Problem Reports. Activities include:	Scripting and Software
Software Configuration Management Process	15	7.2.3 Problem Reporting, Tracking, and Corrective Action	c. Problem Reports that require corrective action of the software product or outputs of software life cycle processes should invoke the change control activity. Activities include:	Manual and Software
Software Configuration Management Process	16	7.2.4 Change Control	a. Change control should preserve the integrity of the configuration items and baselines by providing protection against their change.	Manual

Software Configuration Management Process	17	7.2.4 Change Control	Activities include: b. Change control should ensure that any change to a configuration item requires a change to its configuration identification.		Manual
Software Configuration Management Process	18	7.2.4 Change Control	Activities include: c. Changes to baselines and to configuration items under change control to produce derivative baselines should be recorded, approved, and tracked. Problem reporting is related to change control, since resolution of a reported problem may result in changes to configuration items or baselines.		Pipeline/Tests
Software Configuration Management Process	19	7.2.4 Change Control	Activities include: d. Software changes should be traced to their origin and the software life cycle processes repeated from the point at which the change affects their outputs. For example, an error discovered at hardware/software integration, that is shown to result from an incorrect design, should result in design correction, code correction, and repetition of the associated integral process activities. It is generally recognized that early implementation of change control assists the control and management of software life cycle process activities.		Software and Pipeline/Tests
Software Configuration Management Process	20	7.2.4 Change Control	Activities include: e. Throughout the change activity, software life cycle data affected by the change should be updated and records should be maintained for the change control activity.		Scripting
Software Configuration Management Process	21	7.2.5 Change Review	Activities include: a. Assessment of the impact of the problem or proposed change on system requirements. Feedback should be provided to the system processes, including the system safety assessment process, and any responses from the system processes should be assessed.		Manual
Software Configuration Management Process	22	7.2.5 Change Review	Activities include: b. Assessment of the impact of the problem or proposed change on software life cycle data identifying changes to be made and actions to be taken.		Manual
Software Configuration Management Process	23	7.2.5 Change Review	Activities include: c. Confirmation that affected configuration items are configuration identified.		Manual
Software Configuration Management Process	24	7.2.5 Change Review	Activities include: d. Feedback of Problem Report or change impact and decisions to affected processes.		Scripting and Manual
Software Configuration Management Process	25	7.2.6 Configuration Status Accounting	Activities include: a. Reporting on configuration item identification, baseline identification, Problem Report status, change history, and release status.		Scripting and Manual
Software Configuration Management Process	26	7.2.6 Configuration Status Accounting	Activities include: b. Definition of the data to be maintained and the means of recording and reporting status of this data.		Manual
Software Configuration Management Process	27	7.2.7 Archive, Retrieval, and Release	Activities include: a. Software life cycle data associated with the software product should be retrievable from the approved source (for example, an archive at the developing organization or company).		Manual and Pipeline/Test
Software Configuration Management Process	28	7.2.7 Archive, Retrieval, and Release	Activities include: b. Procedures should be established to ensure the integrity of the stored data, regardless of medium of storage, by: 1. Ensuring that no unauthorized changes can be made.		Scripting and Manual
Software Configuration Management Process	29	7.2.7 Archive, Retrieval, and Release	Activities include: b. Procedures should be established to ensure the integrity of the stored data, regardless of medium of storage, by: 2. Selecting storage media that minimize regeneration errors or deterioration.		Manual
Software Configuration Management Process	30	7.2.7 Archive, Retrieval, and Release	Activities include: b. Procedures should be established to ensure the integrity of the stored data, regardless of medium of storage, by: 3. Preventing loss or corruption of data over time. Depending on the storage media used, this may include periodically exercising the media or refreshing the archived data.		Manual
Software Configuration Management Process	31	7.2.7 Archive, Retrieval, and Release	Activities include: b. Procedures should be established to ensure the integrity of the stored data, regardless of medium of storage, by: 4. Storing duplicate copies in physically separate archives that minimize the risk of loss in the event of a disaster.		Scripting and Manual
Software Configuration Management Process	32	7.2.7 Archive, Retrieval, and Release	Activities include: c. The duplication process should be verified to produce accurate copies, and procedures should exist that ensure error-free copying of the Executable Object Code and Parameter Data Item Files, if any		Scripting and Manual

Software Configuration Management Process	33	7.2.7 Archive, Retrieval, and Release	Activities include: d. Configuration items should be identified and released prior to use for software manufacture and the authority for their release should be established. As a minimum, the components of the software product loaded into the airborne system or equipment should be released. This includes the Executable Object Code and Parameter Data Item Files, if any, and may also include associated media for software loading.		Manual
Software Configuration Management Process	34	7.2.7 Archive, Retrieval, and Release	Activities include: e. Data retention procedures should be established to satisfy airworthiness requirements and enable software modifications.		Software and Pipeline/Tests
Software Configuration Management Process	35	7.4 Software Load Control	Activities include: a. Procedures for part numbering and media identification that identify software configurations that are intended to be approved for loading into the airborne system or equipment.		Manual
Software Configuration Management Process	36	7.4 Software Load Control	Activities include: b. Whichever method is used, software load control should include: b. Whether the software is delivered as an end item or is delivered installed in the airborne system or equipment, records should be kept that confirm software compatibility with the airborne system or equipment hardware.		Scripting and Manual
Software Configuration Management Process	37	7.5 Software Life Cycle Environment Control	The software life cycle environment tools are defined by the software planning process and identified in the Software Life Cycle Environment Configuration Index (see 11.15). Activities include: a. Configuration identification should be established for the Executable Object Code, or equivalent, of the tools used to develop, control, build, verify, and load the software.		Manual and Software
Software Configuration Management Process	38	7.5 Software Life Cycle Environment Control	The software life cycle environment tools are defined by the software planning process and identified in the Software Life Cycle Environment Configuration Index (see 11.15). Activities include: b. The SCM process for controlling qualified tools should comply with the objectives associated with Control Category 1 or Control Category 2 data (see 7.3), according to the guidance provided by section 12.2.3.		Manual
Software Configuration Management Process	39	7.5 Software Life Cycle Environment Control	The software life cycle environment tools are defined by the software planning process and identified in the Software Life Cycle Environment Configuration Index (see 11.15). Activities include: c. Unless section 7.5b applies, the SCM process for controlling the Executable Object Code, or equivalent, for tools used to build and load the software (for example, compilers, assemblers, and linkage editors) should comply with the objectives associated with Control Category 2 data, as a minimum.		Manual
		Chapter	Requirements		
Software Configuration Index	1	11.16 Software Configuration Index	The Software Configuration Index (SCI) identifies the configuration of the software product. Specific configuration identifiers and version identifiers should be provided. The SCI should identify: a. The software product.		Manual
Software Configuration Index	2	11.16 Software Configuration Index	The Software Configuration Index (SCI) identifies the configuration of the software product. Specific configuration identifiers and version identifiers should be provided. The SCI should identify: b. Executable Object Code and Parameter Data Item Files, if any.		Manual
Software Configuration Index	3	11.16 Software Configuration Index	The Software Configuration Index (SCI) identifies the configuration of the software product. Specific configuration identifiers and version identifiers should be provided. The SCI should identify: c. Each Source Code component.		Software
Software Configuration Index	4	11.16 Software Configuration Index	The Software Configuration Index (SCI) identifies the configuration of the software product. Specific configuration identifiers and version identifiers should be provided. The SCI should identify: d. Previously developed software in the software product, if used.		Software
Software Configuration Index	5	11.16 Software Configuration Index	The Software Configuration Index (SCI) identifies the configuration of the software product. Specific configuration identifiers and version identifiers should be provided. The SCI should identify: e. Software life cycle data.		Manual
Software Configuration Index	6	11.16 Software Configuration Index	The Software Configuration Index (SCI) identifies the configuration of the software product. Specific configuration identifiers and version identifiers should be provided. The SCI should identify: f. Archive and release media.		Manual

Software Configuration Index	7	11.16 Software Configuration Index	The Software Configuration Index (SCI) identifies the configuration of the software product. Specific configuration identifiers and version identifiers should be provided. The SCI should identify: g. Instructions for building the Executable Object Code and Parameter Data Item Files, if any, including, for example, instructions and data for compiling and linking; and the procedures used to recover the software for regeneration, testing, or modification. The Software Configuration Index (SCI) identifies the configuration of the software product. Specific configuration identifiers and version identifiers should be provided. The SCI should identify: h. Reference to the Software Life Cycle Environment Configuration Index (see 11.15), if it is packaged separately.		Manual and Pipeline/Tests
Software Configuration Index	8	11.16 Software Configuration Index	The Software Configuration Index (SCI) identifies the configuration of the software product. Specific configuration identifiers and version identifiers should be provided. The SCI should identify: i. Data integrity checks for the Executable Object Code, if used.		Manual
Software Configuration Index	9	11.16 Software Configuration Index	The Software Configuration Index (SCI) identifies the configuration of the software product. Specific configuration identifiers and version identifiers should be provided. The SCI should identify: j. Procedures, methods, and tools for making modifications to the user-modifiable software, if any.		Pipeline/Tests
Software Configuration Index	10	11.16 Software Configuration Index	The Software Configuration Index (SCI) identifies the configuration of the software product. Specific configuration identifiers and version identifiers should be provided. The SCI should identify: k. Procedures and methods for loading the software into the target hardware.		Manual and Software
Software Configuration Index	11	11.16 Software Configuration Index			Software
		Chapter	Requirements		
Software Configuration Management Plan	1	11.4 Software Configuration Management Plan	This plan should include: a. Environment: A description of the SCM environment to be used, including procedures, tools, methods, standards, organizational responsibilities, and interfaces.		Manual
Software Configuration Management Plan	2	11.4 Software Configuration Management Plan	This plan should include: b. Activities: A description of the SCM process activities in the software life cycle: 1. Configuration identification: Items to be identified, when they will be identified, the identification methods for software life cycle data (for example, part numbering), and the relationship of software identification and the system or equipment identification.		Manual
Software Configuration Management Plan	3	11.4 Software Configuration Management Plan	This plan should include: b. Activities: A description of the SCM process activities in the software life cycle: 2. Baselines and traceability: The means of establishing baselines, what baselines will be established, when these baselines will be established, the software library controls, and the configuration item and baseline traceability.		Manual
Software Configuration Management Plan	4	11.4 Software Configuration Management Plan	This plan should include: b. Activities: A description of the SCM process activities in the software life cycle: 3. Problem reporting: The content and identification of Problem Reports for the software product and software life cycle processes, when they will be written, the method of closing Problem Reports, and the relationship to the change control activity.		Manual
Software Configuration Management Plan	5	11.4 Software Configuration Management Plan	This plan should include: b. Activities: A description of the SCM process activities in the software life cycle: 4. Change control: Configuration items and baselines to be controlled, when they will be controlled, the problem/change control activities that control them, pre-certification controls, post-certification controls, and the means of preserving the integrity of baselines and configuration items.		Scripting and Manual
Software Configuration Management Plan	6	11.4 Software Configuration Management Plan	This plan should include: b. Activities: A description of the SCM process activities in the software life cycle: 5. Change review: The method of handling feedback from and to the software life cycle processes; the methods of assessing and prioritizing problems, approving changes, and handling their resolution or change implementation; and the relationship of these methods to the problem reporting and change control activities.		Scripting
Software Configuration Management Plan	7	11.4 Software Configuration Management Plan	This plan should include: b. Activities: A description of the SCM process activities in the software life cycle: 6. Configuration status accounting: The data to be recorded to enable reporting configuration management status, definition of where that data will be kept, how it will be retrieved for reporting, and when it will be available.		Manual

Software Configuration Management Plan	8	11.4 Software Configuration Management Plan	This plan should include: b. Activities: A description of the SCM process activities in the software life cycle: 7. Archive, retrieval and release: The integrity controls, the release method and authority, and data retention.		Manual and Pipeline/Tests
Software Configuration Management Plan	9	11.4 Software Configuration Management Plan	This plan should include: b. Activities: A description of the SCM process activities in the software life cycle: 8. Software load control: A description of the software load control safeguards and records.		Manual and Software
Software Configuration Management Plan	10	11.4 Software Configuration Management Plan	This plan should include: b. Activities: A description of the SCM process activities in the software life cycle: 9. Software life cycle environment controls: Controls for the tools used to develop, build, verify, and load the software, addressing sections 11.4.b.1 through 11.4.b.7. This includes control of tools to be qualified.		Manual
Software Configuration Management Plan	11	11.4 Software Configuration Management Plan	This plan should include: b. Activities: A description of the SCM process activities in the software life cycle: 10. Software life cycle data controls: Controls associated with CC1 and CC2 data.		Manual
Software Configuration Management Plan	12	11.4 Software Configuration Management Plan	c. Transition criteria: The transition criteria for entering the SCM process.		Manual
Software Configuration Management Plan	13	11.4 Software Configuration Management Plan	d. SCM data: A definition of the software life cycle data produced by the SCM process, including SCM Records, the Software Configuration Index, and the Software Life Cycle Environment Configuration Index.		Manual
Software Configuration Management Plan	14	11.4 Software Configuration Management Plan	e. Supplier control: The means of applying SCM process requirements to suppliers.		Manual
		Chapter	Requirements		
Software Environment Configuration Index	1	11.15 Software Life Cycle Environment Configuration Index	This index is written to aid reproduction of the hardware and software life cycle environment for software regeneration, reversion, or software modification, and should: a. Identify the software life cycle environment hardware and its operating system software.		Manual and Software
Software Environment Configuration Index	2	11.15 Software Life Cycle Environment Configuration Index	This index is written to aid reproduction of the hardware and software life cycle environment for software regeneration, reversion, or software modification, and should: b. Identify the tools to be used during the development of the software. Examples include compilers, linkage editors, loaders, data integrity tools such as tools that calculate and embed checksums or cyclical redundancy checks, and any autocode generator with its associated options.		Manual
Software Environment Configuration Index	3	11.15 Software Life Cycle Environment Configuration Index	This index is written to aid reproduction of the hardware and software life cycle environment for software regeneration, reversion, or software modification, and should: c. Identify the test environment used to verify the software product, for example, the software testing and analysis tools.		Manual and Pipeline/Tests
Software Environment Configuration Index	4	11.15 Software Life Cycle Environment Configuration Index	This index is written to aid reproduction of the hardware and software life cycle environment for software regeneration, reversion, or software modification, and should: d. Identify qualified tools and their associated tool qualification data.		Manual

		Chapter	Requirements		Automation Type
Software Quality Assurance Plan	1	11.5 Software Quality Assurance Plan	This plan should include: a. Environment: A description of the SQA environment, including scope, organizational responsibilities and interfaces, standards, procedures, tools, and methods.		Manual
Software Quality Assurance Plan	2	11.3 Software Verification Plan	This plan should include: b. Authority: A statement of the SQA authority, responsibility, and independence, including the SQA approval of software products.		Manual
Software Quality Assurance Plan	3	11.3 Software Verification Plan	This plan should include: c. Activities: The SQA activities that are to be performed for each software life cycle process and throughout the software life cycle including: 1. SQA methods, for example, reviews, audits, reporting, inspections, and monitoring of the software life cycle processes.		Manual and Software
Software Quality Assurance Plan	4	11.3 Software Verification Plan	This plan should include: c. Activities: The SQA activities that are to be performed for each software life cycle process and throughout the software life cycle including: 2. Activities related to the problem reporting, tracking, and corrective action system.		Manual and Software
Software Quality Assurance Plan	5	11.3 Software Verification Plan	This plan should include: c. Activities: The SQA activities that are to be performed for each software life cycle process and throughout the software life cycle including: 3. A description of the software conformity review activity.		Manual
Software Quality Assurance Plan	6	11.3 Software Verification Plan	This plan should include: d. Transition criteria: The transition criteria for entering the SQA process.		Scripting and Manual
Software Quality Assurance Plan	7	11.3 Software Verification Plan	This plan should include: e. Timing: The timing of the SQA process activities in relation to the activities of the software life cycle processes.		Manual
Software Quality Assurance Plan	8	11.3 Software Verification Plan	This plan should include: f. SQA Records: A definition of the records to be produced by the SQA process.		Manual
Software Quality Assurance Plan	9	11.3 Software Verification Plan	This plan should include: g. Supplier oversight: A description of the means of ensuring that suppliers' processes and outputs will comply with the plans and standards.		Manual
		Chapter	Requirements		
Software Quality Assurance Process	1	8.2 Software Quality Assurance Process Activities	Activities for satisfying the SQA process objectives include: a. The SQA process should take an active role in the activities of the software life cycle processes, and have those performing the SQA process enabled with the authority, responsibility, and independence to ensure that the SQA process objectives are satisfied.		Manual and Software
Software Quality Assurance Process	2	8.2 Software Quality Assurance Process Activities	Activities for satisfying the SQA process objectives include: b. The SQA process should provide assurance that software plans and standards are developed and reviewed for compliance with this document and for consistency.		Scripting, Manual and Software
Software Quality Assurance Process	3	8.2 Software Quality Assurance Process Activities	Activities for satisfying the SQA process objectives include: c. The SQA process should provide assurance that the software life cycle processes comply with the approved software plans and standards.		Scripting, Manual and Software
Software Quality Assurance Process	4	8.2 Software Quality Assurance Process Activities	Activities for satisfying the SQA process objectives include: d. The SQA process should include audits of the software life cycle processes during the software life cycle to obtain assurance that: 1. Software plans are available as specified in section 4.2.		Scripting, Manual and Software
Software Quality Assurance Process	5	8.2 Software Quality Assurance Process Activities	Activities for satisfying the SQA process objectives include: d. The SQA process should include audits of the software life cycle processes during the software life cycle to obtain assurance that: 2. Deviations from the software plans and standards are detected, recorded, evaluated, tracked, and resolved.		Scripting, Manual, Software and Pipeline/Tests
Software Quality Assurance Process	6	8.2 Software Quality Assurance Process Activities	Activities for satisfying the SQA process objectives include: d. The SQA process should include audits of the software life cycle processes during the software life cycle to obtain assurance that: 3. Approved deviations are recorded. : It is generally accepted that early detection of process deviations assists efficient achievement of software life cycle process objectives.		Manual and Software

Software Quality Assurance Process	7	8.2 Software Quality Assurance Process Activities	Activities for satisfying the SQA process objectives include: d. The SQA process should include audits of the software life cycle processes during the software life cycle to obtain assurance that: 4. The software development environment has been provided as specified in the software plans.		Manual and Software
Software Quality Assurance Process	8	8.2 Software Quality Assurance Process Activities	Activities for satisfying the SQA process objectives include: d. The SQA process should include audits of the software life cycle processes during the software life cycle to obtain assurance that: 5. The problem reporting, tracking, and corrective action process activities comply with the Software Configuration Management Plan.		Scripting, Manual and Software
Software Quality Assurance Process	9	8.2 Software Quality Assurance Process Activities	Activities for satisfying the SQA process objectives include: d. The SQA process should include audits of the software life cycle processes during the software life cycle to obtain assurance that: 6. Inputs provided to the software life cycle processes by the system processes, including the system safety assessment process, have been addressed.		Scripting, Manual and Software
Software Quality Assurance Process	10	8.2 Software Quality Assurance Process Activities	Activities for satisfying the SQA process objectives include: e. The SQA process should provide assurance that the transition criteria for the software life cycle processes have been satisfied in compliance with the approved software plans. : Monitoring of the activities of software life cycle processes may be performed to provide assurance that the activities are under control.		Manual, Software and Pipeline/Tests
Software Quality Assurance Process	11	8.2 Software Quality Assurance Process Activities	Activities for satisfying the SQA process objectives include: f. The SQA process should provide assurance that software life cycle data is controlled in accordance with the control categories as defined in section 7.3 and the tables of Annex A.		Manual and Software
Software Quality Assurance Process	12	8.2 Software Quality Assurance Process Activities	Activities for satisfying the SQA process objectives include: g. Prior to the delivery of software products submitted as part of a certification application, a software conformity review should be conducted.		Manual, Software and Pipeline/Tests
Software Quality Assurance Process	13	8.2 Software Quality Assurance Process Activities	Activities for satisfying the SQA process objectives include: h. The SQA process should produce records of the SQA process activities (see 11.19), including audit results and evidence of completion of the software conformity review for each software product submitted as part of certification application.		Scripting, Software and Pipeline/Tests
Software Quality Assurance Process	14	8.2 Software Quality Assurance Process Activities	Activities for satisfying the SQA process objectives include: i. The SQA process should provide assurance that supplier processes and outputs comply with approved software plans and standards.		Manual and Software
Software Quality Assurance Process	15	8.3 Software Conformity Review	This review should determine that: a. Planned software life cycle process activities for certification credit, including the generation of software life cycle data, have been completed and records of their completion are retained.		Scripting, Manual and Software
Software Quality Assurance Process	16	8.3 Software Conformity Review	This review should determine that: b. Software life cycle data developed from specific system requirements, safety-related requirements, or software requirements are traceable to those requirements.		Software and Pipeline/Tests
Software Quality Assurance Process	17	8.3 Software Conformity Review	This review should determine that: c. Evidence exists that software life cycle data have been produced in accordance with software plans and standards, and is controlled in accordance with the SCM Plan.		Scripting and Manual
Software Quality Assurance Process	18	8.3 Software Conformity Review	This review should determine that: d. Evidence exists that Problem Reports have been evaluated and have their status recorded.		Scripting and Manual
Software Quality Assurance Process	19	8.3 Software Conformity Review	This review should determine that: e. Software requirement deviations are recorded and approved.		Manual
Software Quality Assurance Process	20	8.3 Software Conformity Review	This review should determine that: f. The Executable Object Code and Parameter Data Item Files, if any, can be regenerated from the archived Source Code.		Software and Pipeline/Tests
Software Quality Assurance Process	21	8.3 Software Conformity Review	This review should determine that: g. The approved software can be loaded successfully through the use of released instructions.		Software
Software Quality Assurance Process	22	8.3 Software Conformity Review	This review should determine that: h. Problem Reports deferred from a previous software conformity review are re-evaluated to determine their status.		Manual

Software Quality Assurance Process	23	8.3 Software Conformity Review	This review should determine that: i. If certification credit is sought for the use of previously developed software, the current software product baseline is traceable to the previous baseline and the approved changes to that baseline.	Manual and Pipeline/Tests
------------------------------------	----	--------------------------------	--	---------------------------

		Chapter	Requirements		Automation Type
Software Requirements Standards	1	11.6 Software Requirements Standards	These standards should include: a. The methods to be used for developing software requirements, such as structured methods.		Manual and Software
Software Requirements Standards	2	11.6 Software Requirements Standards	These standards should include: b. Notations to be used to express requirements, such as data flow diagrams and formal specification languages.		Manual
Software Requirements Standards	3	11.6 Software Requirements Standards	These standards should include: c. Constraints on the use of the tools used for requirements development.		Manual
Software Requirements Standards	4	11.6 Software Requirements Standards	These standards should include: d. The method to be used to provide derived requirements to the system processes.		Manual
		Chapter	Requirements		
Software Requirements Process	1	5.1.2 Software Requirements Process Activities	The software requirements process is complete when its objectives and the objectives of the integral processes associated with it are satisfied. Activities for this process include: a. The system functional and interface requirements that are allocated to software should be analyzed for ambiguities, inconsistencies, and undefined conditions.		Scripting and Manual
Software Requirements Process	2	5.1.2 Software Requirements Process Activities	The software requirements process is complete when its objectives and the objectives of the integral processes associated with it are satisfied. Activities for this process include: b. Inputs to the software requirements process detected as inadequate or incorrect should be reported as feedback to the input source processes for clarification or correction.		Scripting and Manual
Software Requirements Process	3	5.1.2 Software Requirements Process Activities	The software requirements process is complete when its objectives and the objectives of the integral processes associated with it are satisfied. Activities for this process include: c. Each system requirement that is allocated to software should be specified in the high-level requirements.		Manual
Software Requirements Process	4	5.1.2 Software Requirements Process Activities	The software requirements process is complete when its objectives and the objectives of the integral processes associated with it are satisfied. Activities for this process include: d. High-level requirements that address system requirements allocated to software to preclude system hazards should be defined.		Manual
Software Requirements Process	5	5.1.2 Software Requirements Process Activities	The software requirements process is complete when its objectives and the objectives of the integral processes associated with it are satisfied. Activities for this process include: e. The high-level requirements should conform to the Software Requirements Standards, and be verifiable and consistent.		Scripting and Manual
Software Requirements Process	6	5.1.2 Software Requirements Process Activities	The software requirements process is complete when its objectives and the objectives of the integral processes associated with it are satisfied. Activities for this process include: f. The high-level requirements should be stated in quantitative terms with tolerances where applicable.		Scripting and Manual

Software Requirements Process	7	5.1.2 Software Requirements Process Activities	The software requirements process is complete when its objectives and the objectives of the integral processes associated with it are satisfied. Activities for this process include: g. The high-level requirements should not describe design or verification detail except for specified and justified design constraints.		Manual
Software Requirements Process	8	5.1.2 Software Requirements Process Activities	The software requirements process is complete when its objectives and the objectives of the integral processes associated with it are satisfied. Activities for this process include: h. Derived high-level requirements and the reason for their existence should be defined.		Manual
Software Requirements Process	9	5.1.2 Software Requirements Process Activities	The software requirements process is complete when its objectives and the objectives of the integral processes associated with it are satisfied. Activities for this process include: i. Derived high-level requirements should be provided to the system processes, including the system safety assessment process.		Manual
Software Requirements Process	10	5.1.2 Software Requirements Process Activities	The software requirements process is complete when its objectives and the objectives of the integral processes associated with it are satisfied. Activities for this process include: j. If parameter data items are planned, the high-level requirements should describe how any parameter data item is used by the software. The high-level requirements should also specify their structure, the attributes for each of their data elements, and, when applicable, the value of each element. The values of the parameter data item elements should be consistent with the structure of the parameter data item and the attributes of its data elements.		Scripting and Manual
	Nr.	Chapter	Requirements		Scripting (File)
Software Requirements Data	1	11.9 Software Requirements Data	Software Requirements Data is a definition of the high-level requirements including the derived requirements. This data should include: a. Description of the allocation of system requirements to software, with attention to safety-related requirements and potential failure conditions.		Manual
Software Requirements Data	2	11.9 Software Requirements Data	Software Requirements Data is a definition of the high-level requirements including the derived requirements. This data should include: b. Functional and operational requirements under each mode of operation.		Manual
Software Requirements Data	3	11.9 Software Requirements Data	Software Requirements Data is a definition of the high-level requirements including the derived requirements. This data should include: c. Performance criteria, for example, precision and accuracy.		Manual
Software Requirements Data	4	11.9 Software Requirements Data	Software Requirements Data is a definition of the high-level requirements including the derived requirements. This data should include: d. Timing requirements and constraints.		Manual
Software Requirements Data	5	11.9 Software Requirements Data	Software Requirements Data is a definition of the high-level requirements including the derived requirements. This data should include: e. Memory size constraints.		Manual
Software Requirements Data	6	11.9 Software Requirements Data	Software Requirements Data is a definition of the high-level requirements including the derived requirements. This data should include: f. Hardware and software interfaces, for example, protocols, formats, frequency of inputs, and frequency of outputs.		Manual
Software Requirements Data	7	11.9 Software Requirements Data	Software Requirements Data is a definition of the high-level requirements including the derived requirements. This data should include: g. Failure detection and safety monitoring requirements.		Manual and Software
Software Requirements Data	8	11.9 Software Requirements Data	Software Requirements Data is a definition of the high-level requirements including the derived requirements. This data should include: h. Partitioning requirements allocated to software, how the partitioned software components interact with each other, and the software level(s) of each partition.		Manual and Software

		Chapter	Requirements		Automation Type
Software Design Standards	1	11.7 Software Design Standards	These Standards should include: a. Design description method(s) to be used.		Manual
Software Design Standards	2	11.7 Software Design Standards	These Standards should include: b. Naming conventions to be used.		Manual
Software Design Standards	3	11.7 Software Design Standards	These Standards should include: c. Conditions imposed on permitted design methods, for example, scheduling, and the use of interrupts and event-driven architectures, dynamic tasking, re-entry, global data, and exception handling, and rationale for their use.		Manual and Software
Software Design Standards	4	11.7 Software Design Standards	These Standards should include: d. Constraints on the use of the design tools.		Software
Software Design Standards	5	11.7 Software Design Standards	These Standards should include: e. Constraints on design, for example, exclusion of recursion, dynamic objects, data aliases, and compacted expressions.		Software
Software Design Standards	6	11.7 Software Design Standards	These Standards should include: f. Complexity restrictions, for example, maximum level of nested calls or conditional structures, use of unconditional branches, and number of entry/exit points of code components.		Software
		Chapter	Requirements		
Software Design Process	1	5.2.2 Software Design Process Activities	The software design process is complete when its objectives and the objectives of the integral processes associated with it are satisfied. Activities for this process include: a. Low-level requirements and software architecture developed during the software design process should conform to the Software Design Standards and be traceable, verifiable, and consistent.		Manual, Pipeline/Tests
Software Design Process	2	5.2.2 Software Design Process Activities	The software design process is complete when its objectives and the objectives of the integral processes associated with it are satisfied. Activities for this process include: b. Derived low-level requirements and the reason for their existence should be defined and analyzed to ensure that the higher level requirements are not compromised.		Manual, Pipeline/Tests
Software Design Process	3	5.2.2 Software Design Process Activities	The software design process is complete when its objectives and the objectives of the integral processes associated with it are satisfied. Activities for this process include: c. Software design process activities could introduce possible modes of failure into the software or, conversely, preclude others. The use of partitioning or other architectural means in the software design may alter the software level assignment for some components of the software. In such cases, additional data should be defined as derived requirements and provided to the system processes, including the system safety assessment process.		Manual and Software
Software Design Process	4	5.2.2 Software Design Process Activities	The software design process is complete when its objectives and the objectives of the integral processes associated with it are satisfied. Activities for this process include: d. Interfaces between software components, in the form of data flow and control flow, should be defined to be consistent between the components.		Software
Software Design Process	5	5.2.2 Software Design Process Activities	The software design process is complete when its objectives and the objectives of the integral processes associated with it are satisfied. Activities for this process include: e. Control flow and data flow should be monitored when safety-related requirements dictate, for example, watchdog timers, reasonableness-checks, and cross-channel comparisons.		Manual and Software
Software Design Process	6	5.2.2 Software Design Process Activities	The software design process is complete when its objectives and the objectives of the integral processes associated with it are satisfied. Activities for this process include: f. Responses to failure conditions should be consistent with the safety-related requirements.		Software
Software Design Process	7	5.2.2 Software Design Process Activities	The software design process is complete when its objectives and the objectives of the integral processes associated with it are satisfied. Activities for this process include: g. Inadequate or incorrect inputs detected during the software design process should be provided to the system life cycle processes, the software requirements process, or the software planning process as feedback for clarification or correction.		Scripting and Software
Software Design Process	8	5.2.3 Designing for User-Modifiable Software	The activities for user-modifiable software include: a. The non-modifiable component should be protected from the modifiable component to prevent interference in the safe operation of the non-modifiable component. This protection can be enforced by hardware, by software, by the tools used to make the change, or by a combination of the three. If the protection is provided by software, it should be designed and verified at the same software level as the non-modifiable software. If the protection is provided by a tool, the tool should be categorized and qualified as defined in section 12.2.		Software

Software Design Process	9	5.2.3 Designing for User-Modifiable Software	The activities for user-modifiable software include: b. The means provided to change the modifiable component should be shown to be the only means by which the modifiable component can be changed.		Software
Software Design Process	10	5.2.4 Designing for Deactivated Code	The activities for deactivated code include: a. A mechanism should be designed and implemented to assure that deactivated functions or components have no adverse impact on active functions or components.		Scripting and Software
Software Design Process	11	5.2.4 Designing for Deactivated Code	The activities for deactivated code include: b. Evidence should be available that the deactivated code is disabled for the environments where its use is not intended. Unintended execution of deactivated code due to abnormal system conditions is the same as unintended execution of activated code.		Manual and Software
Software Design Process	12	5.2.4 Designing for Deactivated Code	The activities for deactivated code include: c. The development of deactivated code, like the development of the active code, should comply with the objectives of this document.		Software
		Chapter	Requirements		
Design Description	1	11.10 Design Description	The Design Description is a definition of the software architecture and the low-level requirements that will satisfy the high-level requirements. This data should include: a. A detailed description of how the software satisfies the specified high-level requirements, including algorithms, data structures, and how software requirements are allocated to processors and tasks.		Manual
Design Description	2	11.10 Design Description	The Design Description is a definition of the software architecture and the low-level requirements that will satisfy the high-level requirements. This data should include: b. The description of the software architecture defining the software structure to implement the requirements.		Manual
Design Description	3	11.10 Design Description	The Design Description is a definition of the software architecture and the low-level requirements that will satisfy the high-level requirements. This data should include: c. The input/output description, for example, a data dictionary, both internally and externally throughout the software architecture.		Manual
Design Description	4	11.10 Design Description	The Design Description is a definition of the software architecture and the low-level requirements that will satisfy the high-level requirements. This data should include: d. The data flow and control flow of the design.		Manual
Design Description	5	11.10 Design Description	The Design Description is a definition of the software architecture and the low-level requirements that will satisfy the high-level requirements. This data should include: e. Resource limitations, the strategy for managing each resource and its limitations, the margins, and the method for measuring those margins, for example, timing and memory.		Manual
Design Description	6	11.10 Design Description	The Design Description is a definition of the software architecture and the low-level requirements that will satisfy the high-level requirements. This data should include: f. Scheduling procedures and inter-processor/inter-task communication mechanisms, including time-rigid sequencing, preemptive scheduling, Ada rendezvous, and interrupts.		Manual
Design Description	7	11.10 Design Description	The Design Description is a definition of the software architecture and the low-level requirements that will satisfy the high-level requirements. This data should include: g. Design methods and details for their implementation, for example, software loading, user-modifiable software, or multiple-version dissimilar software.		Manual and Software
Design Description	8	11.10 Design Description	The Design Description is a definition of the software architecture and the low-level requirements that will satisfy the high-level requirements. This data should include: h. Partitioning methods and means of preventing partition breaches.		Manual
Design Description	9	11.10 Design Description	The Design Description is a definition of the software architecture and the low-level requirements that will satisfy the high-level requirements. This data should include: i. Descriptions of the software components, whether they are new or previously developed, and, if previously developed, reference to the baseline from which they were taken.		Manual
Design Description	10	11.10 Design Description	The Design Description is a definition of the software architecture and the low-level requirements that will satisfy the high-level requirements. This data should include: j. Derived requirements resulting from the software design process.		Manual
Design Description	11	11.10 Design Description	The Design Description is a definition of the software architecture and the low-level requirements that will satisfy the high-level requirements. This data should include: k. If the system contains deactivated code, a description of the means to ensure that the code cannot be enabled in the target computer.		Manual

Design Description	12	11.10 Design Description	The Design Description is a definition of the software architecture and the low-level requirements that will satisfy the high-level requirements. This data should include: I. Rationale for those design decisions that are traceable to safety-related system requirements.		Scripting
--------------------	----	--------------------------	--	--	-----------

		Chapter	Requirements		Automation Type
Trace Data	1	11.21 Trace Data	Trace Data establishes the associations between life cycle data items contents. Trace Data should be provided that demonstrates bi-directional associations between: a. System requirements allocated to software and high-level requirements.		Manual
Trace Data	2	11.21 Trace Data	Trace Data establishes the associations between life cycle data items contents. Trace Data should be provided that demonstrates bi-directional associations between: b. High-level requirements and low-level requirements.		Manual
Trace Data	3	11.21 Trace Data	Trace Data establishes the associations between life cycle data items contents. Trace Data should be provided that demonstrates bi-directional associations between: c. Low-level requirements and Source Code.		Manual and Software
Trace Data	4	11.21 Trace Data	Trace Data establishes the associations between life cycle data items contents. Trace Data should be provided that demonstrates bi-directional associations between: d. Software Requirements and test cases.		Manual and Pipeline/Tests
Trace Data	5	11.21 Trace Data	Trace Data establishes the associations between life cycle data items contents. Trace Data should be provided that demonstrates bi-directional associations between: e. Test cases and test procedures.		Pipeline/Tests
Trace Data	6	11.21 Trace Data	Trace Data establishes the associations between life cycle data items contents. Trace Data should be provided that demonstrates bi-directional associations between: f. Test procedures and test results.		Pipeline/Tests
		Chapter	Requirements		
Problem Reports	1	11.17 Problem Reports	Problem Reports should include: a. Identification of the configuration item and/or the software life cycle process activity in which the problem was observed.		Scripting and Manual
Problem Reports	2	11.17 Problem Reports	Problem Reports should include: b. Identification of the configuration item(s) to be modified or a description of the process to be changed.		Scripting and Manual
Problem Reports	3	11.17 Problem Reports	Problem Reports should include: c. A problem description that enables the problem to be understood and resolved. The problem description should contain sufficient detail to facilitate the assessment of the potential safety or functional effects of the problem.		Scripting and Manual
Problem Reports	4	11.17 Problem Reports	Problem Reports should include: d. A description of the corrective action taken to resolve the reported problem.		Manual

		Chapter	Requirements		Automation Type
Certification Liaison Process	1	9.1 Means of Compliance and Planning	Activities include: a. Submitting the Plan for Software Aspects of Certification and other requested data to the certification authority for review.		Scripting and Manual
Certification Liaison Process	2	9.1 Means of Compliance and Planning	Activities include: b. Resolving issues identified by the certification authority concerning the planning for the software aspects of certification.		Manual
Certification Liaison Process	3	9.1 Means of Compliance and Planning	Activities include: c. Obtaining agreement with the certification authority on the Plan for Software Aspects of Certification.		Manual
Certification Liaison Process	4	9.2 Compliance Substantiation	Activities include: a. Resolving issues raised by the certification authority as a result of its reviews.		Manual and Software
Certification Liaison Process	5	9.2 Compliance Substantiation	Activities include: b. Submitting the Software Accomplishment Summary (see 11.20) and Software Configuration Index (see 11.16) to the certification authority.		Scripting and Manual
Certification Liaison Process	6	9.2 Compliance Substantiation	Activities include: c. Submitting or making available other data or evidence of compliance requested by the certification authority.		Scripting and Manual

		Chapter	Requirements	Automation Type
Software Planning Process	1	4.2 Software Planning Process Activities	Activities for the software planning process include: a. The software plans should be developed that provide direction to the personnel performing the software life cycle processes. See also 9.1.	Manual
Software Planning Process	2	4.2 Software Planning Process Activities	Activities for the software planning process include: b. The software development standards to be used for the project should be defined or selected.	Manual and Software
Software Planning Process	3	4.2 Software Planning Process Activities	Activities for the software planning process include: c. Methods and tools should be chosen that aid error prevention and provide defect detection in the software development processes.	Manual and Software
Software Planning Process	4	4.2 Software Planning Process Activities	Activities for the software planning process include: d. The software planning process should provide coordination between the software development and integral processes to provide consistency among strategies in the software plans.	Manual and Software
Software Planning Process	5	4.2 Software Planning Process Activities	Activities for the software planning process include: e. The means should be specified to revise the software plans as a project progresses.	Manual
Software Planning Process	6	4.2 Software Planning Process Activities	Activities for the software planning process include: f. When multiple-version dissimilar software is used in a system, the software planning process should choose the methods and tools to achieve the dissimilarity necessary to satisfy the system safety objectives.	Manual and Software
Software Planning Process	7	4.2 Software Planning Process Activities	Activities for the software planning process include: g. For the software planning process to be complete, the software plans and software development standards should be under change control and reviews of them completed (see 4.6).	Software and Pipeline/Tests
Software Planning Process	8	4.2 Software Planning Process Activities	Activities for the software planning process include: h. If deactivated code is planned, the software planning process should describe how the deactivation mechanism and deactivated code will be defined and verified to satisfy system safety objectives.	Manual and Software
Software Planning Process	9	4.2 Software Planning Process Activities	Activities for the software planning process include: i. If user-modifiable software is planned, related processes, tools, environment, and data items substantiating the design (see 5.2.3) should be specified in the software plans and standards.	Manual and Software
Software Planning Process	10	4.2 Software Planning Process Activities	Activities for the software planning process include: j. When parameter data items are planned, the following should be addressed: 1. The way that parameter data items are used.	Manual and Software
Software Planning Process	11	4.2 Software Planning Process Activities	Activities for the software planning process include: j. When parameter data items are planned, the following should be addressed: 2. The software level of the parameter data items.	Manual and Software
Software Planning Process	12	4.2 Software Planning Process Activities	Activities for the software planning process include: j. When parameter data items are planned, the following should be addressed: 3. The processes to develop, verify, and modify parameter data items, and any associated tool qualification.	Manual and Software
Software Planning Process	13	4.2 Software Planning Process Activities	Activities for the software planning process include: j. When parameter data items are planned, the following should be addressed: 4. Software load control and compatibility.	Manual and Software
Software Planning Process	14	4.2 Software Planning Process Activities	Activities for the software planning process include: k. The software planning process should address any additional considerations that are applicable.	Manual and Software
Software Planning Process	15	4.2 Software Planning Process Activities	Activities for the software planning process include: l. If software development activities will be performed by a supplier, planning should address supplier oversight.	Manual
Software Planning Process	16	4.3 Software Plans	Activities for the software plans include: a. The software plans should comply with this document. (RTCA DO178C)	Manual
Software Planning Process	17	4.3 Software Plans	Activities for the software plans include: b. The software plans should define the transition criteria for software life cycle processes by specifying: 1. The inputs to the process, including feedback from other processes.	Manual and Software
Software Planning Process	18	4.3 Software Plans	Activities for the software plans include: b. The software plans should define the transition criteria for software life cycle processes by specifying: 2. Any integral process activities that may be required to act on these inputs.	Manual and Software
Software Planning Process	19	4.3 Software Plans	Activities for the software plans include: b. The software plans should define the transition criteria for software life cycle processes by specifying: 3. Availability of tools, methods, plans, and procedures.	Manual and Software
Software Planning Process	20	4.3 Software Plans	Activities for the software plans include: c. The software plans should state the procedures to be used to implement software changes prior to use on a certified product. Such changes may be as a result of feedback from other processes and may cause a change to the software plans.	Manual and Software
Software Planning Process	21	4.4.1 Software Development Environment	Activities for the selection of software development environment methods and tools include: a. During the software planning process, the software development environment should be chosen to reduce its potential risk to the software being developed.	Manual and Software

Software Planning Process	22	4.4.1 Software Development Environment	Activities for the selection of software development environment methods and tools include: b. The use of tools or combinations of tools and parts of the software development environment should be chosen to achieve the necessary level of confidence that an error introduced by one part would be detected by another. An acceptable environment is produced when both parts are consistently used together. This selection includes the assessment of the need for tool qualification.	Manual and Software
Software Planning Process	23	4.4.1 Software Development Environment	Activities for the selection of software development environment methods and tools include: c. The software verification process activities or software development standards, which include consideration of the software level, should be defined to reduce potential software development environment-related errors.	Manual, Software and Pipeline/Tests
Software Planning Process	24	4.4.1 Software Development Environment	Activities for the selection of software development environment methods and tools include: d. If certification credit is sought for use of the tools in combination, the sequence of operation of the tools should be specified in the appropriate plan.	Manual and Software
Software Planning Process	25	4.4.1 Software Development Environment	Activities for the selection of software development environment methods and tools include: e. If optional features of software tools are chosen for use in a project, the effects of the options should be examined and specified in the appropriate plan. This is especially important for compilers and autocode generators.	Manual and Software
Software Planning Process	26	4.4.1 Software Development Environment	Activities for the selection of software development environment methods and tools include: f. Known tool problems and limitations should be assessed and those issues which can adversely affect airborne software should be addressed.	Manual and Software
Software Planning Process	27	4.4.2 Language and Compiler Considerations	The software planning process considers these features when choosing a programming language and planning for verification. Activities include: a. Some compilers have features intended to optimize performance of the object code. If the test cases give coverage consistent with the software level, the correctness of the optimization need not be verified. Otherwise, the impact of these features on structural coverage analysis should be determined. Additional information can be found in section 6.4.4.2.	Manual and Software
Software Planning Process	28	4.4.2 Language and Compiler Considerations	The software planning process considers these features when choosing a programming language and planning for verification. Activities include: b. To implement certain features, compilers for some languages may produce object code that is not directly traceable to the Source Code, for example, initialization, built-in error detection, or exception handling (see 6.4.4.2 b). The software planning process should provide a means to detect this object code and to ensure verification coverage, and should define the means in the appropriate plan.	Manual, Software and Pipeline/Tests
Software Planning Process	29	4.4.2 Language and Compiler Considerations	The software planning process considers these features when choosing a programming language and planning for verification. Activities include: c. If a new compiler, linkage editor, or loader version is introduced, or compiler options are changed during the software life cycle, previous tests and coverage analyses may no longer be valid. The verification planning should provide a means of reverification that is consistent with sections 6 and 12.1.3.	Manual and Pipeline/Tests
Software Planning Process	30	4.4.3 Software Test Environment	Activities include: a. The emulator or simulator may need to be qualified as described in section 12.2.	Manual and Software
Software Planning Process	31	4.4.3 Software Test Environment	Activities include: b. The differences between the target computer and the emulator or simulator, and the effects of these differences on the ability to detect errors and verify functionality, should be considered. Detection of those errors should be provided by the software verification process and specified in the Software Verification Plan.	Manual, Software and Pipeline/Tests
Software Planning Process	32	4.5 Software Development Standards	Activities for development of the software standards include: a. The software development standards should comply with section 11.	Manual and Software
Software Planning Process	33	4.5 Software Development Standards	Activities for development of the software standards include: b. The software development standards should enable software components of a given software product or related set of products to be uniformly designed and implemented.	Manual and Software
Software Planning Process	34	4.5 Software Development Standards	Activities for development of the software standards include: c. The software development standards should disallow the use of constructs or methods that produce outputs that cannot be verified or that are not compatible with safety-related requirements.	Manual and Software
Software Planning Process	35	4.5 Software Development Standards	Activities for development of the software standards include: d. Robustness should be considered in the software development standards.	Manual and Software
Software Planning Process	36	4.6 Review of the Software Planning Process	Activities include: a. Methods are chosen that enable the objectives of this document to be satisfied.	Manual and Software
Software Planning Process	37	4.6 Review of the Software Planning Process	Activities include: b. Software life cycle processes can be applied consistently.	Manual and Software
Software Planning Process	38	4.6 Review of the Software Planning Process	Activities include: c. Each process produces evidence that its outputs can be traced to their activity and inputs, showing the degree of independence of the activity, the environment, and the methods to be used.	Manual and Pipeline/Tests
Software Planning Process	39	4.6 Review of the Software Planning Process	Activities include: d. The outputs of the software planning process are consistent and comply with section 11.	Manual and Software
		Chapter	Requirements	

Software Development Plan	1	11.2 Software Development Plan	Standards. Identification of the Software Requirements Standards, Software Design Standards, and Software Code Standards for the project. Also, references to the standards for previously developed software, including COTS software, if those standards are different.		Manual
Software Development Plan	2	11.2 Software Development Plan	Software life cycle. A description of the software life cycle processes to be used to form the specific software life cycle(s) to be used on the project, including the transition criteria for the software development processes. This description is distinct from the summary provided in the Plan for Software Aspects of Certification, in that it provides the detail necessary to ensure proper implementation of the software life cycle processes.		Manual and Software
Software Development Plan	3	11.2 Software Development Plan	Software development environment. A statement of the chosen software development environment in terms of hardware and software, including: 1. The requirements development method(s) and tools to be used.		Manual
Software Development Plan	4	11.2 Software Development Plan	Software development environment. A statement of the chosen software development environment in terms of hardware and software, including: 2. The design method(s) and tools to be used.		Manual
Software Development Plan	5	11.2 Software Development Plan	Software development environment. A statement of the chosen software development environment in terms of hardware and software, including: 3. The coding method(s), programming language(s), coding tool(s) to be used, and when applicable, options and constraints of autocode generators.		Manual and Software
Software Development Plan	6	11.2 Software Development Plan	Software development environment. A statement of the chosen software development environment in terms of hardware and software, including: 4. The compilers, linkage editors, and loaders to be used.		Manual and Pipeline/Tests
Software Development Plan	7	11.2 Software Development Plan	Software development environment. A statement of the chosen software development environment in terms of hardware and software, including: 5. The hardware platforms for the tools to be used.		Manual
		Chapter	Requirements		
Plan for Software Aspects of Certification	1	11.1 Plan for software Aspects of Certification	System overview. This section provides an overview of the system, including a description of its functions and their allocation to the hardware and software, the architecture, processor(s) used, hardware/software interfaces, and safety features.		Manual and Software
Plan for Software Aspects of Certification	2	11.1 Plan for software Aspects of Certification	Software overview. This section briefly describes the software functions with emphasis on the proposed safety and partitioning concepts. Examples include resource sharing, redundancy, fault tolerance, mitigation of single event upset, and timing and scheduling strategies.		Manual and Software
Plan for Software Aspects of Certification	3	11.1 Plan for software Aspects of Certification	Certification considerations. This section provides a summary of the certification basis, including the means of compliance, as relating to the software aspects of certification. This section also states the proposed software level(s) and summarizes the justification provided by the system safety assessment process, including potential software contributions to failure conditions.		Manual
Plan for Software Aspects of Certification	4	11.1 Plan for software Aspects of Certification	Software life cycle. This section defines the software life cycle to be used and includes a summary of each of the software life cycle processes for which detailed information is defined in their respective software plans. The summary explains how the objectives of each software life cycle process will be satisfied, and specifies the organizations to be involved, the organizational responsibilities, and the system life cycle processes and certification liaison process responsibilities.		Manual
Plan for Software Aspects of Certification	5	11.1 Plan for software Aspects of Certification	Software life cycle data. This section specifies the software life cycle data that will be produced and controlled by the software life cycle processes. This section also describes the relationship of the data to each other or to other data defining the system, the software life cycle data to be submitted to the certification authority, the form of the data, and the means by which the data will be made available to the certification authority.		Manual and Software
Plan for Software Aspects of Certification	6	11.1 Plan for software Aspects of Certification	Schedule. This section describes the means the applicant will use to provide the certification authority with visibility of the activities of the software life cycle processes so reviews can be planned.		Manual

Plan for Software Aspects of Certification	7	11.1 Plan for software Aspects of Certification	This plan should include: Additional considerations: This section describes specific considerations that may affect the certification process. Examples include alternative methods of compliance, tool qualification, previously developed software, option-selectable software, user-modifiable software, deactivated code, COTS software, field-loadable software, parameter data items, multiple-version dissimilar software, and product service history.		Manual
Plan for Software Aspects of Certification	8	11.1 Plan for software Aspects of Certification	This plan should include: Supplier oversight. This section describes the means of ensuring that supplier processes and outputs will comply with approved software plans and standards.		Manual

		Chapter	Requirements		Automation Type
Software Code Standards	1	11.8 Software Code Standards	These Standards should include: a. Programming language(s) to be used and/or defined subset(s). For a programming language, reference the data that unambiguously defines the syntax, the control behavior, the data behavior, and side-effects of the language. This may require limiting the use of some features of a language. These Standards should include:		Manual and Software
Software Code Standards	2	11.8 Software Code Standards	b. Source Code presentation standards, for example, line length restriction, indentation, and blank line usage and Source Code documentation standards, for example, name of author, revision history, inputs and outputs, and affected global data. These Standards should include:		Manual and Software
Software Code Standards	3	11.8 Software Code Standards	c. Naming conventions for components, subprograms, variables, and constants. These Standards should include:		Manual
Software Code Standards	4	11.8 Software Code Standards	d. Conditions and constraints imposed on permitted coding conventions, such as the degree of coupling between software components and the complexity of logical or numerical expressions and rationale for their use. These Standards should include:		Manual
Software Code Standards	5	11.8 Software Code Standards	e. Constraints on the use of the coding tools.		Manual and Software
		Chapter	Requirements		
Software Coding Process	1	5.3.2 Software Coding Process Activities	The software coding process is complete when its objectives and the objectives of the integral processes associated with it are satisfied. Activities for this process include: a. The Source Code should implement the low-level requirements and conform to the software architecture.		Software
Software Coding Process	2	5.3.2 Software Coding Process Activities	The software coding process is complete when its objectives and the objectives of the integral processes associated with it are satisfied. Activities for this process include: b. The Source Code should conform to the Software Code Standards.		Software
Software Coding Process	3	5.3.2 Software Coding Process Activities	The software coding process is complete when its objectives and the objectives of the integral processes associated with it are satisfied. Activities for this process include: c. Inadequate or incorrect inputs detected during the software coding process should be provided to the software requirements process, software design process, and/or software planning process as feedback for clarification or correction.		Scripting and Software
Software Coding Process	4	5.3.2 Software Coding Process Activities	The software coding process is complete when its objectives and the objectives of the integral processes associated with it are satisfied. Activities for this process include: d. Use of autocode generators should conform to the constraints defined in the planning process.		Pipeline/Tests

		Chapter	Requirements		Automation Type
Integration Process	1	5.4.2 Integration Process Activities	The integration process is complete when its objectives and the objectives of the integral processes associated with it are satisfied. Activities for this process include: a. The object code and Executable Object Code should be generated from the Source Code and compiling, linking, and loading data. Any Parameter Data Item File should be generated.		Software and Pipeline/Tests
Integration Process	2	5.4.2 Integration Process Activities	The integration process is complete when its objectives and the objectives of the integral processes associated with it are satisfied. Activities for this process include: b. Software integration should be performed on a host computer, a target computer emulator, or the target computer.		Software
Integration Process	3	5.4.2 Integration Process Activities	The integration process is complete when its objectives and the objectives of the integral processes associated with it are satisfied. Activities for this process include: c. The software should be loaded into the target computer for hardware/software integration.		Software
Integration Process	4	5.4.2 Integration Process Activities	The integration process is complete when its objectives and the objectives of the integral processes associated with it are satisfied. Activities for this process include: d. Inadequate or incorrect inputs detected during the integration process should be provided to the software requirements process, the software design process, the software coding process, or the software planning process as feedback for clarification or correction.		Scripting and Software
Integration Process	5	5.4.2 Integration Process Activities	The integration process is complete when its objectives and the objectives of the integral processes associated with it are satisfied. Activities for this process include: e. Patches should not be used in software submitted for use in a certified product to implement changes in requirements or architecture, or changes found necessary as a result of the software verification process activities. Patches may be used on a limited, case-by-case basis, for example, to resolve known deficiencies in the software development environment such as a known compiler problem.		Manual
Integration Process	6	5.4.2 Integration Process Activities	The integration process is complete when its objectives and the objectives of the integral processes associated with it are satisfied. Activities for this process include: f. When a patch is used, these should be available: 1. Confirmation that the software configuration management process can effectively track the patch.		Manual and Pipeline/Tests
Integration Process	7	5.4.2 Integration Process Activities	The integration process is complete when its objectives and the objectives of the integral processes associated with it are satisfied. Activities for this process include: f. When a patch is used, these should be available: 2. An analysis to provide evidence that the patched software satisfies all the applicable objectives.		Manual and Pipeline/Tests
Integration Process	8	5.4.2 Integration Process Activities	The integration process is complete when its objectives and the objectives of the integral processes associated with it are satisfied. Activities for this process include: f. When a patch is used, these should be available: 3. Justification in the Software Accomplishment Summary for use of a patch.		Manual

Appendix B

SCIR Template

Software Configuration Index

Reference ID:	Package-Name
Generation Date:	dd/mm/yyyy - hh:mm:ss
#nb of errors	xx
#nb of Warnings	xx

SCI Errors

Error description...
Error description...
Error description...
...

SCI Warnings

Warning description...
Warning description...
...

SCI Data

Specification Stream					
Filename	VCS Commit / Version	VCS Location	Jira ID	Lifecycle	Responsible
ITEM-NAME_SYS-DESC.docx	https://../projects/DOCOPS/repos/fake-csw/commits/e406e9eb2409bd4bd9c305e6a395415e41451973	https://../projects/DOCOPS/repos/fake-csw/browse/moduleX	ABC-XXX (open/ done/ in prog]	SYS-DESC	John John (username)

ITEM-NAME_SYS-SPEC.docx	https://../projects/DOCOPS/repos/fake-csw/commits/e406e9eb2409bd4bd9c305e6a395415e41451973	https://../projects/DOCOPS/repos/fake-csw/browse/moduleX	ABC-XXX (open/ done/ in prog)	SYS-SPEC	John John (username)
ITEM-NAME_SYS-SPEC-RVW.docx	https://../projects/DOCOPS/repos/fake-csw/commits/e406e9eb2409bd4bd9c305e6a395415e41451973	https://../projects/DOCOPS/repos/fake-csw/browse/moduleX	ABC-XXX (open/ done/ in prog)	SYS-SPEC	Andy Peter (username)

Software Implementation Stream					
Filename	VCS Commit / Version	VCS Location	Jira ID	Lifecycle	Responsible
ITEM-NAME_CODE.c	https://../projects/DOCOPS/repos/fake-csw/commits/e406e9eb2409bd4bd9c305e6a395415e41451973	https://../projects/DOCOPS/repos/fake-csw/browse/moduleX	ABC-XXX (open/ done/ in prog)	SYS-DESC	Andy Peter (username)
ITEM-NAME_CODE-RVW.docx	https://../projects/DOCOPS/repos/fake-csw/commits/e406e9eb2409bd4bd9c305e6a395415e41451973	https://../projects/DOCOPS/repos/fake-csw/browse/moduleX	ABC-XXX (open/ done/ in prog)	SYS-DESC	John John (username)
ITEM-NAME_HLR-SPEC.docx	https://../projects/DOCOPS/repos/fake-csw/commits/e406e9eb2409bd4bd9c305e6a395415e41451973	https://../projects/DOCOPS/repos/fake-csw/browse/moduleX	ABC-XXX (open/ done/ in prog)	SYS-DESC	Andy Peter (username)

Software Testing Stream					
Filename	VCS Commit / Version	VCS Location	Jira ID	Lifecycle	Responsible
...	...	...	...	...	...

HIS & System Testing Stream					
Filename	VCS Commit / Version	VCS Location	Jira ID	Lifecycle	Responsible
...	...	...	...	...	...
...	...	...	...	...	...

Miscellaneous Artefacts					
Filename	VCS Commit / Version	VCS Location	Jira ID	Lifecycle	Responsible
...	...	...	...	Misc	...
...	...	...	...	Misc	...

Appendix C

Generated SCIR

The following file was generated using a mock repository with the following files (1) SpecStage1.docx, (2) SpecStage2.pdf, (3) ImplStage1.zip, (4) ImplStage2.docx, (5) TestStage1.pdf, and (6) RandomFile.docx.

Software Configuration Index Report

Generation Date:	23/06/2022 14:57:37
#nb of errors:	1
#nb of Warnings:	0

SCI Errors

--

SCI Warnings

Software Testing Stage has no associated artifacts
--

Specification Stage

Filename	VCS PR / Version	VCS Location	Lifecycle	Responsible
SpecStage1.docx	https://api.github.com/repos/jmscrodrigues/fake_csw/pulls/1	https://github.com/jmscrodrigues/fake_csw/SpecStage1.docx	SpecStage1	Joaquim Rodrigues
SpecStage2.pdf	https://api.github.com/repos/jmscrodrigues/fake_csw/pulls/1	https://github.com/jmscrodrigues/fake_csw/SpecStage1.docx	SpecStage2	Joaquim Rodrigues

Implementation Stage

Filename	VCS PR / Version	VCS Location	Lifecycle	Responsible
ImplStage1.zip	https://api.github.com/repos/jmscrodrigues/fake_csw/pulls/2	https://github.com/jmscrodrigues/fake_csw/ImplStage1.zip	ImplStage1	Joaquim Rodrigues
ImplStage2.docx	https://api.github.com/repos/jmscrodrigues/fake_csw/pulls/2	https://github.com/jmscrodrigues/fake_csw/ImplStage2.docx	ImplStage2	Joaquim Rodrigues

Testing Stage

Filename	VCS PR / Version	VCS Location	Lifecycle	Responsible
HSI and System Testing Stage				

Filename	VCS PR / Version	VCS Location	Lifecycle	Responsible
TestStage1.pdf	https://api.github.com/repos/jmscrodrigues/fake_csw/pulls/3	https://github.com/jmscrodrigues/fake_csw/TestStage1.pdf	TestStage1	Joaquim Rodrigues

Miscellaneous Artifacts				
Filename	VCS PR / Version	VCS Location	Lifecycle	Responsible
RandomFile.docx	https://api.github.com/repos/jmscrodrigues/fake_csw/pulls/4	https://github.com/jmscrodrigues/fake_csw/TestStage1.pdf	RandomFile	Joaquim Rodrigues

Appendix D

Document Traceability Tool - Log File

The following figure reflects a log file when executing the Document Traceability Automation Tool when missing (1) PPR file and (2) D00007300 file.

```
[2022-06-23 16:05:07.502649] : Warning: File PPR not present in the bundle  
[2022-06-23 16:05:07.503611] : Warning: File D00007300 not present in the bundle
```


Appendix E

CSW Interview Guide

Improving Agility in Critical Software Systems Development

Este questionário tem o intuito de avaliar o contributo, importância e performance de duas ferramentas (SCIR Generator e Document Verifier) desenvolvidas no trabalho de uma dissertação assim como a sua aplicabilidade para o contexto do desenvolvimento de sistemas críticos.

**Obrigatório*

1. Tempo de experiência (em anos) *

2. Duração (estimada) para o preenchimento manual de um Software Configuration Index Report (SCIR), em horas *

3. Opinião acerca da inclusão de um SCIR Generator na pipeline *

Marcar apenas uma oval.

12345

Pouco interessante/benéficoMuito interessante/benéfico

4. Duração (estimada) para a verificação da presença de todos os ficheiros no data package a ser entregue à autoridade certificadora, em horas *

5. Opinião acerca da inclusão de um verificador de ficheiros automático na pipeline *

Marcar apenas uma oval.

	1	2	3	4	5	
Pouco interessante/benéfico	☐	☐	☐	☐	☐	Muito interessante/benéfico

6. Sugestões de processos a automatizar no que toca à produção e gestão de documentação certificável *

Este conteúdo não foi criado nem aprovado pela Google.

Google Formulários