Capítulo 3

Fundamentals of IEC 62304 with an Agile Software Development Model

Johnny Marques, Lilian Barros, Sarasuaty Yelisetty, Talita Slavov

Abstract

In 2006, a working group from the International Electrotechnical Commission (IEC) defined IEC 62304:2006. Thus, the use of IEC 62304 has become fully harmonized in the United States and Europe. The purpose of IEC 62304 is to provide requirements for Healthcare Systems manufacturers with Software to demonstrate their ability to provide Software that consistently meets customer requirements and applicable regulatory requirements. Objective: This chapter aims to present the fundamentals of IEC 62304 with an Agile Software Development Model. Justification and motivation: In 2021, IEC 62304 completes 15 years. Thus, the authors believe that there are already works that report experiences, analyzes and difficulties in its use. These results can be interesting to direct further research and definitions of methods, models, guides or other materials to comply with IEC 62304. Conclusion: This book chapter provided a lecture with fundamentals of IEC 62304, including an Agile Software Development Model.

3.1. Introduction

The standards published by committees, international technical entities or regulatory agencies influence the development of Software in Regulated Environments through guidelines for Software processes and products [Munch et al. 2012], considering the risk mentioned above. In addition, there are several similarities between the standards for software development in the aviation, health, and railway areas [Marques and Cunha 2019].

Faced with the challenges that involve the insertion of informatics in the health field, specialized organizations in the area decided to produce guidelines and policies that can help in the process of implantation and evaluation of systems with software [Mauer and Marin 2017].

A Medical System (MS) is composed of several physical and logical parts. One or more Medical Devices (MDs) are part of a Medical System (MS), and they are responsible for the necessary information controls [Marques et al. 2021]. A Medical Device Software

(MDS) is loaded and operated into a Medical Device (MD). Each MDS is composed of Software Items (SIs), which are any identifiable part of a computer program. According to Magnusson (2012), all Medical Devices need to meet the regulations to ensure the safety of the user and the patient. In addition, with the increased use of Software on these systems, entities such as the Food and Drug Administration (FDA) in the United States have identified the need for specific regulation.

In 2006, a working group from the International Electrotechnical Commission (IEC) defined IEC 62304:2006 [IEC 2006]. Thus, the use of IEC 62304 has become fully harmonized in the United States and Europe. The purpose of IEC 62304 is to provide requirements for Healthcare Systems manufacturers with Software to demonstrate their ability to provide Software that consistently meets customer requirements and applicable regulatory requirements.

Figure 3.1 presents an illustrative association of these concepts. IEC 62304 operates within the scope of the MDS, SIs and Software Units (SUs). The IEC 62304:2006 [IEC 2006] and its amendment IEC 62304:2006/AMD 1:2015 [IEC 2015] present the requirements associated with the rigor of the MDS development and maintenance processes, according to the safety risk [Marques 2019].

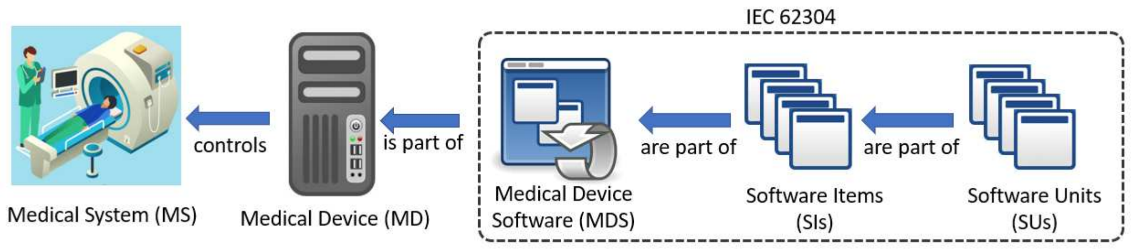


Figura 3.1. Scope of IEC 62304 [Marques et al. 2021]

In 2021, IEC 62304 completes 15 years. Thus, the authors believe that there are already works that report experiences, analyzes and difficulties in its use. These results can be interesting to direct further research and definitions of methods, models, guides or other materials to comply with IEC 62304. Therefore, this chapter aims to present the fundamentals of IEC 62304 with an Agile Software Development Model.

In addition to this section 1, this book chapter has another 11 (eleven) sections. Section 2 presents the acronyms, and section 3 presents the definitions. Section 4 presents the general requirements. Section 5 describes the software development process. Section 6 presents the software maintenance process. Section 7 presents the software risk process. Section 8 describes the software configuration management process. Section 9 presents the software problem resolution process. Section 10 briefly describes the relationship between IEC 62304 and other standards. Section 11 presents an overview of agile methods and Scrum. Section 12 describes our Agile Software Development Model. Section 13 presents the related work identified during a Systematic Literature Mapping. Finally, section 14 presents the final considerations.

3.2. Acronyms

Table 3.1 presents the acronyms required for this book chapter.

Tabela 3.1. Acronyms

Acronym	Definition
ASD	Adaptive Software Development
Cat	Categories
CMMI	Capability Maturity Model Integration
Co	Contribution
CR	Change Request
DSDM	Dynamic Software Development Method
FDA	Food and Drug Administration
FDD	Feature Driven Development
IEC	International Electrotechnical Commission
ISO	International Organization for Standardization
MAD	Manifesto for Agile Development
MD	Medical Device
MDS	Medical Device Software
MS	Medical System
PR	Problem Report
SDP	Software Development Plan
SI	Software Item
SLM	Systematic Literature Mapping
SOUP	Software of Unknown Provenance
SU	Software Unit
TDD	Test Driven Development
Var	Variability
XP	Extreme Programming

3.3. Definitions

Table 3.2 presents some definitions required for this book chapter.

3.4. General Requirements

The IEC 62304 defines the life cycle requirements for MDS. The structure of the standard follows a hierarchy composed by processes, activities, and tasks, establishing a common framework for MDS life cycle processes. The hierarchy is illustrated as part of Figure 3.2.

Compliance with the IEC 62304 is defined as implementing all the processes, activities, and tasks identified in this standard following the software safety class. In addition, the MDS manufacturer must identify the safety risks to users (patients and medical staff) regarding software misbehavior. These risks are directly correlated with the software safety classes defined by IEC 62304.

Tabela 3.2. Terms and definitions

Term	Definition
Activity	A set of one or more interrelated or interacting tasks.
Anomaly	Any condition that deviates from the expected based on requirements specifications, design documents, standards, or from someone's perceptions or experiences.
Configuration item	Entity that can be uniquely identified at a given reference point.
Legacy software	Medical Device Software which was legally placed on the market with insufficient objective evidence that it was developed in compliance with the current version of this standard.
Manufacturer	Organization, which conducts software development, maintenance, and other processes, activities, and tasks in the scope of IEC 62304.
Problem report	A record of the actual or potential behavior of the MDS that a user or other interested person believes to be unsafe, inappropriate for the intended use or contrary to specification.
Process	A set of interrelated or interacting activities that transform inputs into outputs.
Risk management file	A set of records and other documents, not necessarily contiguous, that is produced by a Risk assessment process.
Safety	Freedom from unacceptable risk.
Serious injury	Injury or illness that: is life-threatening results in permanent impairment of a body function or permanent damage to a body structure or necessitates medical or surgical intervention.
Software item	Any identifiable part of a computer program.
Software system	Integrated collection of software items organized to accomplish a specific function or set of functions.
Software unit	Software item that is not subdivided into other items.
Task	A single piece of work that needs to be done.

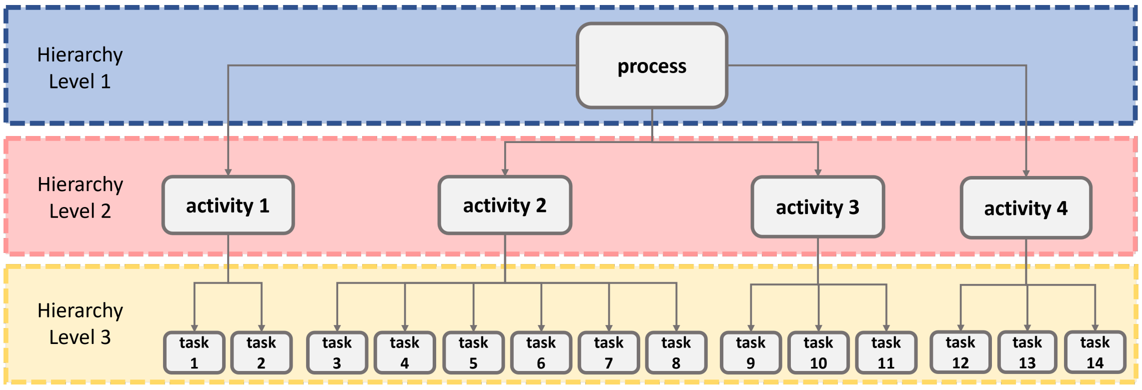


Figura 3.2. Hierarchy of the MDS life cycle

The Medical Device Software manufacturer must apply a risk management process using ISO 14971:2019 [ISO 2019], where safety risks are identified to users (patients and medical staff) regarding software misbehavior. This analysis will define the software classes (A, B and C) provided by IEC 62304. The safety classes determine the rigor in the software development process through the activities. Marques et al. (2021) presented the number of activities associated to each class, as shown in Table 3.3.

Tabela 3.3. Software safety classes, impacts and total of activities required

Class	Impact	Required Tasks
A	No injury or damage to health is possible.	48
B	Non-serious injury is possible.	85
C	Death or serious injury is possible.	89

IEC 62304 had an amendment introduced in 2015 and is under review for improvement. The expectation is that IEC 62304 will have a revision in the year 2021. One of the main contributions of the amendment was the flow of risk classification for the Software Systems existing in a Health System. Initially, all analysis begins with the most severe category (Class C). After following the flow provided in Figure 3.3, the classification can be changed to Class A or B if the conditions existing in the risk analysis are met. The amendment also brought editorial and conceptual changes that needed updating.

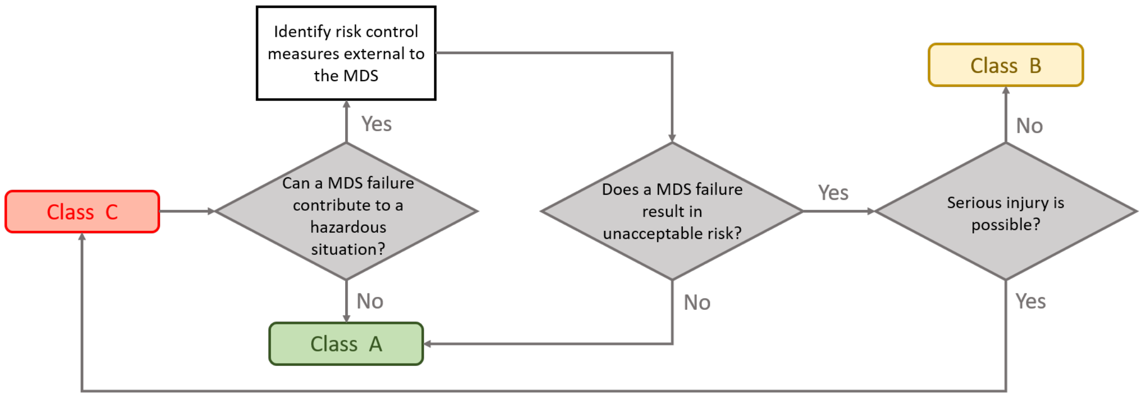

Identify risk control measures external to the MDS
Class B
Yes
No
Class C
Can a MDS failure contribute to a hazardous situation?
Does a MDS failure result in unacceptable risk?
Yes
Serious injury is possible?
No
No
Yes
Class A

Figura 3.3. Assignment of the software class from the safety risk analysis - adapted from [IEC 2015]

IEC 62304 describes 5 (five) processes: Software Development Process, Software Maintenance Process, Software Risk Management Process, Software Configuration Management Process, and Software Problem Resolution Process, as shown in Figure 3.4. The Software Development Process contains 8 (eight) activities.

The Software of Unknown Provenance (SOUP) is a SI that is already developed and generally available and has not been developed to be incorporated a MD or Software previously developed for which adequate records of the development processes are not available.

3.5. Software Development Process

Some software life cycle models were created over the years. Although they define different strategies for executing the software development and verification, it is possible to see

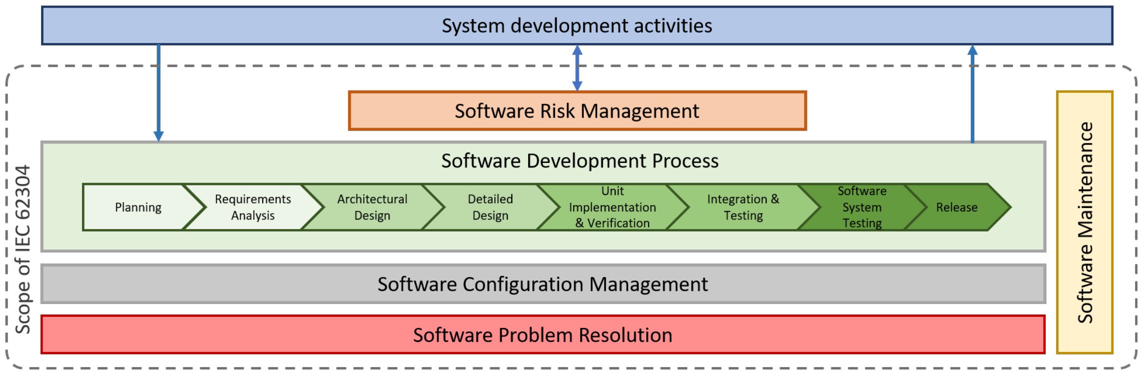


Figura 3.4. Overview of IEC 62304 Processes

that such models differ only in the granularity and the involvement of the user in the evaluation of the software [Sommerville 2015][Pressman and Maxim 2015][Tsui et al. 2015]. The software development is generally divided into 7 (seven) subprocesses: Requirements, Architecture, Design, Implementation, Tests, Verification, and Release [Wasson 2015]. IEC 62304 contains 8 (eight) activities as part of the *Software development process*:

1. *Software development plan*;
2. *Software requirements analysis*;
3. *Software architectural design*;
4. *Software detailed design*;
5. *Software unit implementation and verification*;
6. *Software integration and integration testing*;
7. *Software system testing*; and
8. *Software release*.

3.5.1. Software Development Planning

The *Software development planning* activity contains 11 (eleven) tasks:

1. *Software development plan* (clause 5.1.1);
2. *Keep software development plan updated* (clause 5.1.2);
3. *Software development plan reference to system design and development* (clause 5.1.3);
4. *Software development standards, methods and tools planning* (clause 5.1.4);
5. *Software integration and integration testing planning* (clause 5.1.5);
6. *Software verification planning* (clause 5.1.6);

7. *Software risk management planning* (clause 5.1.7);
8. *Documentation planning* (clause 5.1.8);
9. *Software configuration management planning* (clause 5.1.9);
10. *Supporting items to be controlled* (clause 5.1.10);
11. *Software configuration item control before verification* (clause 5.1.11); and
12. *Identification and avoidance of common software defects* (clause 5.1.12).

Table 3.4 presents the applicability of each task inside the software classes.

Tabela 3.4. Summary of tasks in *Software development planning* activity

Clause	Class A	Class B	Class C
5.1.1	X	X	X
5.1.2	X	X	X
5.1.3	X	X	X
5.1.4			X
5.1.5		X	X
5.1.6	X	X	X
5.1.7	X	X	X
5.1.8	X	X	X
5.1.9	X	X	X
5.1.10		X	X
5.1.11		X	X
5.1.12		X	X

As part of the activity *Software development plan*, the manufacturer shall establish a Software Development Plan (SDP) for conducting the activities of the software development process appropriate to the scope, magnitude, and Software safety classifications of the Software system to be developed. The software development lifecycle shall either be fully defined in the SDP. The plan shall address the following:

1. The processes to be used in the development of the MDS;
2. The artifacts of the activities and tasks;
3. The traceability between System Requirements, Software System Requirements, and System Tests;
4. Software configuration and change management, including SOUP configuration items and software, used to support development; and
5. The software problem resolution for handling problems detected in the deliverables and activities at each stage of the life cycle.

As part of the task *Keep software development plan updated*, the manufacturer shall update the plan as development proceeds as appropriate.

As part of the task *Software development plan reference to system design and development*, System Requirements, as inputs for software development, shall be referenced in the SDP. In addition, the manufacturer shall include or reference the SDP procedures for coordinating the software development and the design and development validation necessary to satisfy customer requirements and applicable regulatory requirements.

As part of the task *Software development standards, methods and tools planning*, the manufacturer shall include or reference in the SDP the standards, methods, and tools associated with the development of Software Items of Class C.

As part of the task *Software integration and integration testing planning*, the manufacturer shall include or reference in the SDP a plan to integrate the SIs (including SOUP) and perform testing during integration. It is acceptable to combine integration testing and Software System testing into a single plan and set of activities.

As part of the task *Software verification planning*, the manufacturer shall include or reference in the software development plan the following verification information:

1. Artifacts requiring verification;
2. The required verification tasks for each lifecycle activity;
3. Milestones at which the artifacts are verified; and
4. The acceptance criteria for each artifact verification.

As part of the task *Software risk management planning*, the manufacturer shall include or reference in the SDP a plan to conduct the activities and tasks of the software Risk Management Process, including the management of risks relating to SOUP.

As part of the task *Documentation planning*, the manufacturer shall include or reference in the SDP information about the documents to be produced during the software development lifecycle. For each identified document or type of document, the following information shall be included or referenced:

1. Title, name or naming convention;
2. Purpose; and
3. Intended audience of the document.

As part of the task *Software configuration management planning*, the manufacturer shall include or reference software configuration management information in the SDP. The software configuration management information shall include or reference:

1. The classes, types, categories or lists of items to be controlled;
2. The software configuration management activities and tasks;

3. The organization(s) responsible for performing software configuration management and activities;
4. Their relationship with other organizations, such as software development or maintenance;
5. When the items are to be placed under configuration control; and
6. When the problem resolution process is to be used.

As part of the task *Supporting items to be controlled*, the items to be controlled shall include tools, items or settings, used to develop and which could impact the MDS.

As part of the task *Software configuration item control before verification*, the manufacturer shall plan to place configuration items under configuration management control before they are verified.

As part of the task *Identification and avoidance of common software defects*, the manufacturer shall include or reference in the SDP a procedure for:

- Identifying categories of defects that may be introduced based on the selected programming technology that are relevant to their Software System; and
- Documenting evidence that demonstrates that these defects do not contribute to unacceptable risk.

3.5.2. Software Requirements Analysis

The *Software requirements analysis* activity captures and defines software requirements from the product level that are implemented by software [Sommerville 2015] [Pressman and Maxim 2015].

The *Software requirements analysis* activity contains 6 (six) tasks:

1. *Define and document software system requirements from system requirements* (clause 5.2.1);
2. *Software system requirements content* (clause 5.2.2);
3. *Include risk control measures in software system requirements* (clause 5.2.3);
4. *Re-evaluate medical device risk analysis* (clause 5.2.4);
5. *Update system requirements* (clause 5.2.5); and
6. *Verify software system requirements* (clause 5.2.6).

Table 3.5 presents the applicability of each task inside the software classes.

As part of the task *Define and document software system requirements from system requirements*, for each MDS, the manufacturer shall define and document Software System Requirements from the System Requirements.

Tabela 3.5. Summary of tasks in *Software requirements analysis* activity

Clause	Class A	Class B	Class C
5.2.1	X	X	X
5.2.2	X	X	X
5.2.3		X	X
5.2.4	X	X	X
5.2.5	X	X	X
5.2.6	X	X	X

As part of the task *Software system requirements content*, the manufacturer shall define and document Software System Requirements from the System Requirements. The manufacturer shall include in the Software System Requirements:

- Functional and capability requirements;
- Software system inputs and outputs;
- Interfaces between the software system and other systems;
- Software-driven alarms, warnings, and operator messages;
- Security requirements;
- User interface requirements implemented by software;
- Data definition and database requirements;
- Installation and acceptance requirements of the delivered MDS at the operation and maintenance site or sites;
- Requirements related to methods of operation and maintenance;
- Requirements related to IT-network aspects;
- User maintenance requirements; and
- Regulatory requirements, according to the nature of the MS.

As part of the task *Include risk control measures in software requirements*, the manufacturer shall include risk control measures implemented in the MDS.

As part of the task *Re-evaluate medical device risk analysis*, the manufacture shall re-evaluate the medical device risk analysis when software requirements are established and update it as appropriate.

As part of the task *Update system requirements*, the manufacturer shall ensure that existing requirements, including System Requirements, are re-evaluated and updated as appropriate as a result of the *Software requirements analysis* activity.

As part of the task *Verify Software System requirements*, the manufacturer shall verify and document that the Software System Requirements:

- Implement System Requirements;
- Do not contradict one another;
- Are expressed in terms that avoid ambiguity;
- Are stated in terms that permit the establishment of test criteria and performance of tests to determine whether the test criteria have been met;
- Can be uniquely identified; and
- Are traceable to System Requirements or another source.

Figure 3.5 presents an overview of the *Software requirements analysis* activity.

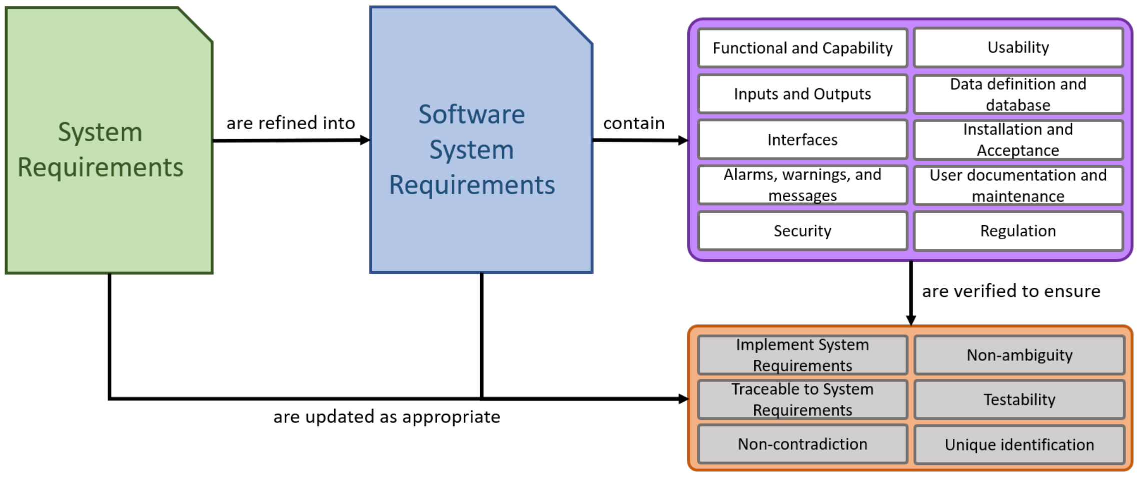

System Requirements
are refined into
Software System Requirements
contain
Functional and Capability
Usability
Inputs and Outputs
Data definition and database
Interfaces
Installation and Acceptance
Alarms, warnings, and messages
User documentation and maintenance
Security
Regulation
are verified to ensure
Implement System Requirements
Non-ambiguity
Traceable to System Requirements
Testability
Non-contradiction
Unique identification
are updated as appropriate

Figura 3.5. Overview of the *Software requirements analysis activity*

3.5.3. Software Architectural Design

The *Software architectural design* activity uses outputs of the software requirements analysis activity to develop the Architecture by creating Software Items and allocating functions expressed by Software System Requirements.

The *Software architectural design* activity contains 6 (six) tasks:

1. *Transform software system requirements into a software architecture* (clause 5.2.1);
2. *Develop an architecture for the interfaces of software items* (clause 5.2.2);
3. *Specify functional and performance requirements of SOUP item* (clause 5.2.3);
4. *Specify system hardware and software required by SOUP item* (clause 5.2.4);
5. *Identify segregation necessary for risk control* (clause 5.2.5); and
6. *Verify software architecture* (clause 5.2.6).

Tabela 3.6. Summary of tasks in *Software architectural design* activity

Clause	Class A	Class B	Class C
5.3.1		X	X
5.3.2		X	X
5.3.3		X	X
5.3.4		X	X
5.3.5			X
5.3.6		X	X

Table 3.6 presents the applicability of each task inside the software classes.

As part of the task *Transform software systems requirements into a software architecture*, the manufacturer transforms the Software System Requirements for the MDS into a documented Architecture that describes the Software's structure and identifies the software items.

As part of the task *Develop a software architecture for the interfaces of software items*, the manufacturer shall develop and document an Architecture for the interfaces between the SIs.

As part of the task *Specify functional and performance requirements of SOUP item*, if a SI is identified as SOUP, the manufacturer shall specify functional and performance requirements for the SOUP item that is necessary for its intended use.

As part of the task *Specify system hardware and Software required by SOUP item*, if a SI is identified as SOUP, the manufacturer shall specify the system hardware and software necessary to support the proper operation of the SOUP item.

As part of the task *Identify segregation necessary for risk control*, the manufacturer shall identify any segregation between SIs that is necessary for risk control, and state how to ensure that such segregation is effective.

As part of the task *Verify software architecture*, the manufacturer verifies and documents that the Architecture of the software implements Software System Requirements and that the Architecture can support interfaces between SIs and between SIs and hardware. The manufacturer shall verify and document that the Architecture:

- Is complete, allocating the Software System Requirements correctly;
- Supports interfaces between Software Items and hardware; and
- Supports the operation of any SOUP items.

Figure 3.6 presents an overview of the *Software architecture design* activity.

3.5.4. Software Detailed Design

The *Software detailed design* activity contains 5 (five) tasks:

1. *Refine software detailed architecture into software units* (clause 5.4.1);

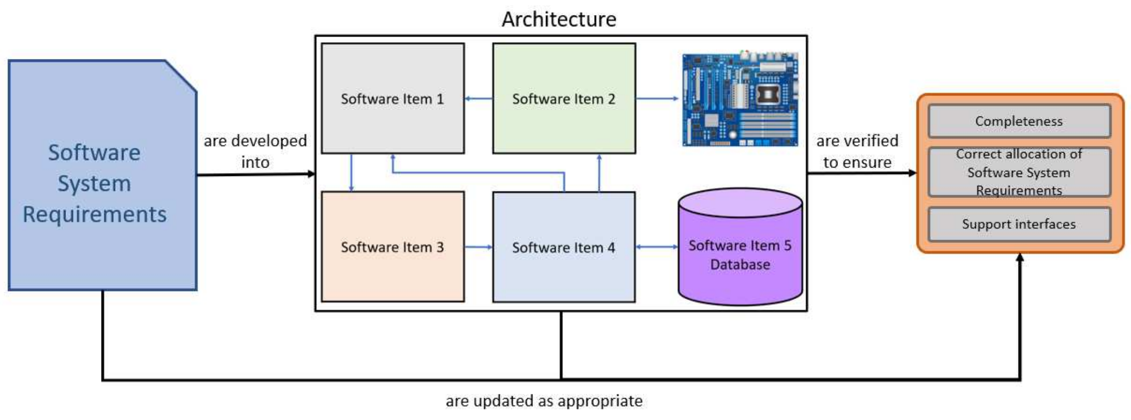


Figura 3.6. Overview of the Software architecture design activity

2. *Develop detailed design for each software unit* (clause 5.4.2);
3. *Develop detailed design for interfaces* (clause 5.4.3); and
4. *Verify detailed design* (clause 5.4.4).

Table 3.7 presents the applicability of each task inside the software classes.

Tabela 3.7. Summary of tasks in *Software detailed design* activity

Clause	Class A	Class B	Class C
5.4.1		X	X
5.4.2			X
5.4.3			X
5.4.4			X

As part of the task *Refine software architecture into software units*, the manufacturer shall refine the Software Architecture until Software Units represent it and develop and document a detailed design for each SU of the SI. The number of levels is defined by the manufacturer, as presented in Figure 3.7.

As part of the task *Develop detailed design for each software unit*, the manufacturer shall document a design with enough detail to allow correct implementation of each SU.

As part of the task *Develop detailed design for interfaces*, the manufacturer shall document a design for any interfaces between the SU and external components (hardware or software), as well as any interfaces between SUs, detailed enough to implement each SU and its interfaces correctly.

As part of the task *Verify detailed design*, the manufacturer shall verify and document that the software detailed design:

- Implements the Architecture;

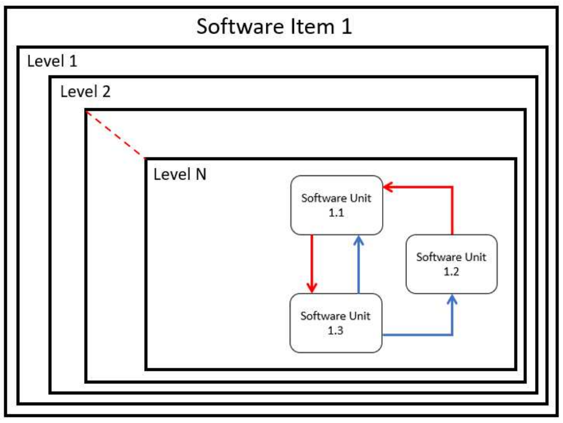


Figura 3.7. Overview of levels of Architecture

- Has correct data and control flow among Software Units;
- Presents the detailed internal logic confirming the refinement from Software System requirements; and
- Is free from contradiction with the Architecture.

Figure 3.8 presents an overview of the *Software detailed design* activity.

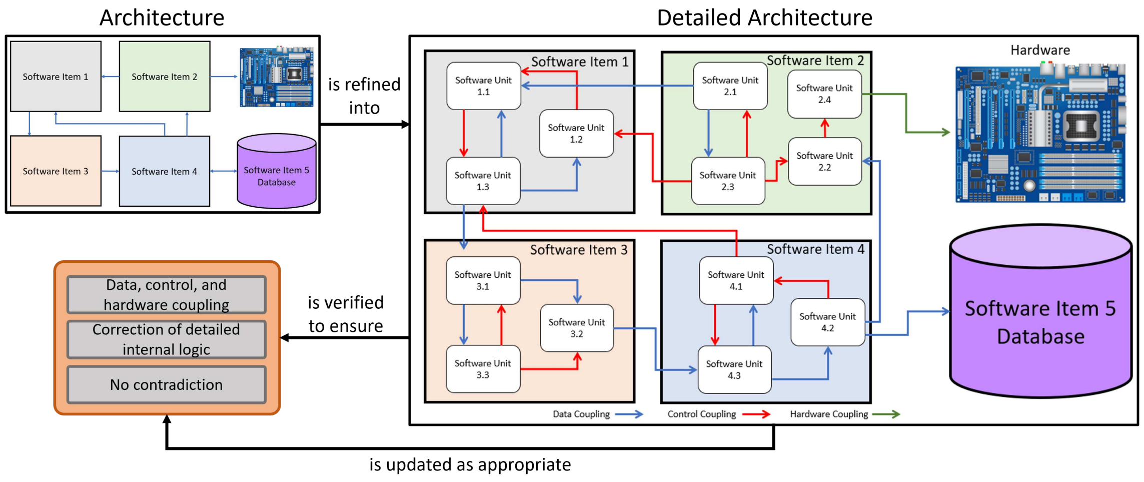


Figura 3.8. Overview of the Software detailed design activity

3.5.5. Software Unit Implementation and Verification

The *Software unit implementation and verification* activity uses outputs of the Software detailed design activity, generating source and executable codes to be loaded inside the selected hardware [Sommerville 2015] [Pressman and Maxim 2015].

The *Software unit implementation and verification* activity contains 5 (five) tasks:

1. *Implement each software unit* (clause 5.5.1);

2. *Establish software unit verification process* (clause 5.5.2);
3. *Software unit acceptance criteria* (clause 5.5.3);
4. *Additional software unit acceptance criteria* (clause 5.5.4); and
5. *Software Unit verification* (clause 5.5.5).

Table 3.8 presents the applicability of each task inside the software classes.

Tabela 3.8. Summary of tasks in *Software unit implementation and verification* activity

Clause	Class A	Class B	Class C
5.5.1	X	X	X
5.5.2		X	X
5.5.3		X	X
5.5.4			X
5.5.5		X	X

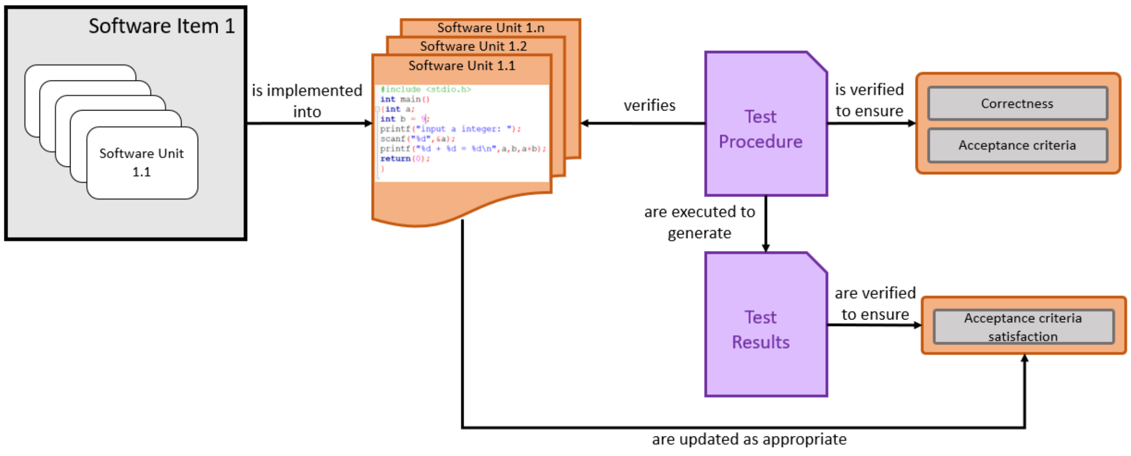
As part of the task *Implement each software unit*, the manufacturer shall implement each Software Unit. In the task *Establish software unit verification process*, the manufacturer shall establish strategies, methods and procedures for verifying each Software Unit is required, where verification is done by testing. The test procedures shall be evaluated for correctness.

As part of the task *Software unit acceptance criteria*, the manufacturer shall establish acceptance criteria for Software Units prior to integration into more oversized Software Items as appropriate and ensure that Software Units meet acceptance criteria.

The manufacturer shall include additional acceptance criteria, according to task *Additional software unit acceptance criteria* as appropriate for:

- Proper event sequence;
- Data and control flow;
- Planned resource allocation;
- Fault handling (error definition, isolation, and recovery);
- Initialization of variables;
- Self-diagnostics;
- Memory management and memory overflows; and
- Boundary conditions.

As part of the task *Software unit verification*, the manufacturer shall execute the test and document the results. Figure 3.9 presents an overview of the *Software unit implementation and verification* activity.



```
#include <stdio.h>
int main()
{int a;
int b = 9;
printf("input a integer: ");
scanf("%d",&a);
printf("%d + %d = %d\n",a,b,a+b);
return(0);
}
```

Figura 3.9. Overview of the *Software unit implementation and verification* activity

3.5.6. Software Integration and Integration Testing

The *Software integration and integration testing* activity contains 8 (eight) tasks:

1. *Integrate software units* (clause 5.6.1);
2. *Verify software integration* (clause 5.6.2);
3. *Software integration testing* (clause 5.6.3);
4. *Software integration testing content* (clause 5.6.4);
5. *Evaluate software integration test procedures* (clause 5.6.5);
6. *Conduct regression tests* (clause 5.6.6);
7. *Integration test record contents* (clause 5.6.7); and
8. *Use software problem resolution process* (clause 5.6.8).

Table 3.9 presents the applicability of each task inside the software classes.

Tabela 3.9. Summary of tasks in *Software integration and integration testing* activity

Clause	Class A	Class B	Class C
5.6.1		X	X
5.6.2		X	X
5.6.3		X	X
5.6.4		X	X
5.6.5		X	X
5.6.6		X	X
5.6.7		X	X
5.6.8		X	X

As part of the task *Integrate software units*, the manufacturer shall integrate the Software Units following the integration plan, ensuring the Software Units have been integrated into Software items and the Software system.

As part of the task *Verify software integration*, the manufacturer shall verify and record the following aspects of the software integration following the integration plan:

1. The SUs have been integrated into SIs and the MDS; and
2. The hardware items, SIs, and support for manual operations (e.g., human-equipment interface, online help menus, speech recognition, voice control) of the MS have been integrated.

As part of the task *Software integration testing*, the manufacturer shall test the integrated software items following the integration plan and document the results. As part of the task *Software integration testing content*, the manufacturer shall address whether the integrated SIs performs as intended. Examples to be considered are:

1. The required functionality of the Software;
2. Specified timing and other behavior;
3. Specified functioning of internal and external interfaces; and
4. Testing under abnormal conditions including foreseeable misuse.

As part of the task *Evaluate software integration test procedures*, the manufacturer shall evaluate the integration Test Procedures for correctness.

As part of the task *Conduct regression tests*, when software items are integrated, the manufacturer shall conduct regression testing appropriate to demonstrate that defects have not been introduced into previously integrated Software.

As part of the task *Integration test record contents*, the manufacturer shall:

- Document the test result (pass/fail and a list of anomalies);
- Retain sufficient records to permit the test to be repeated; and
- Identify the tester.

As part of the task *Use software problem resolution process*, the manufacturer shall enter anomalies found during software integration and integration testing into a software problem resolution process.

Figure 3.10 presents an overview of the *Software integration and integration testing* activity.

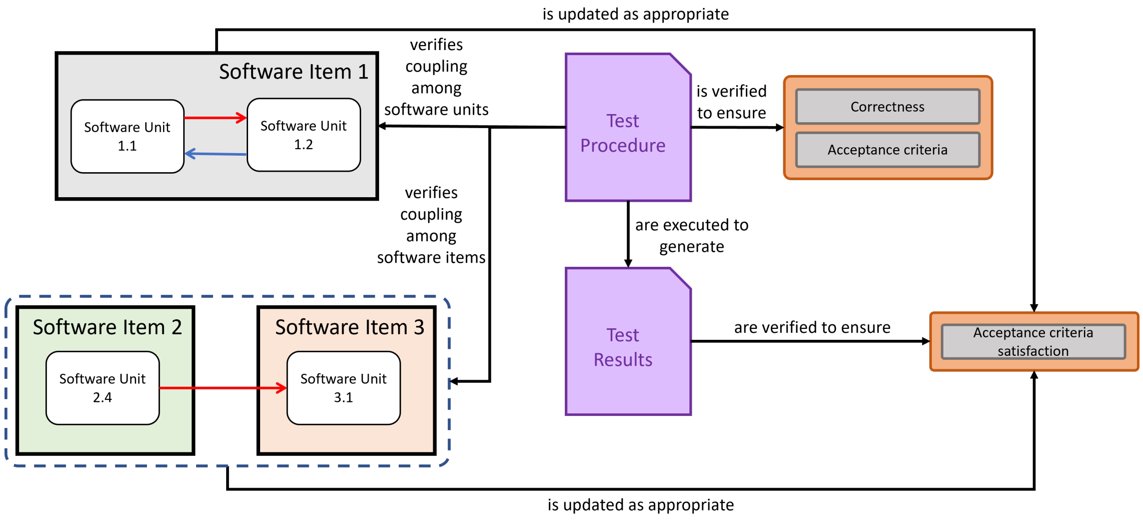


Figura 3.10. Overview of the *Software integration and integration testing* activity

3.5.7. Software System Testing

The *Software system testing* activity contains 5 (five) tasks:

1. *Establish tests for software system requirements* (clause 5.7.1);
2. *Use software problem resolution process* (clause 5.7.2);
3. *Retest after changes* (clause 5.7.3);
4. *Evaluate software system testing* (clause 5.7.4); and
5. *Software system test record contents* (clause 5.7.5).

Table 3.10 presents the applicability of each task inside the software classes.

Tabela 3.10. Summary of tasks in *Software system testing* activity

Clause	Class A	Class B	Class C
5.7.1	X	X	X
5.7.2	X	X	X
5.7.3	X	X	X
5.7.4	X	X	X
5.7.5	X	X	X

As part of the task *Establish tests for software system requirements*, the manufacturer shall establish and perform a set of tests, expressed as input stimuli, expected outcomes, pass/fail criteria and procedures, for conducting Software system testing such that all software requirements are covered. Separate tests for each requirement and tests of combinations of requirements can be performed, primarily if dependencies between requirements exist.

As part of the task *Use software problem resolution process*, the manufacturer shall enter anomalies found during Software System testing into a software problem resolution process.

As part of the task *Retest after changes*, when changes are made during Software system testing, the manufacturer shall:

1. Repeat tests, perform modified tests or perform additional tests, as appropriate, to verify the effectiveness of the change in correcting the problem;
2. Conduct testing appropriate to demonstrate that unintended side effects have not been introduced; and
3. Perform relevant risk management activities.

As part of the task *Verify software system testing*, the manufacturer shall verify that:

- Software System test procedures trace to software requirements;
- All software requirements have been tested or otherwise verified; and
- Test results meet the required pass/fail criteria.

As part of the task *Software system test record contents*, the manufacturer shall:

- A reference to test case procedures showing required actions and expected results;
- The test result (pass/fail and a list of anomalies);
- The version of software tested;
- Relevant hardware and software test configurations;
- Relevant test tools;
- Date tested; and
- The identity of the person responsible for executing the test and recording the test results.

Figure 3.11 presents an overview of the *Software system testing* activity.

3.5.8. Software Release

The *Software release* activity contains 8 (eight) tasks:

1. *Ensure software verification is complete* (clause 5.8.1);
2. *Document known residual anomalies* (clause 5.8.2);

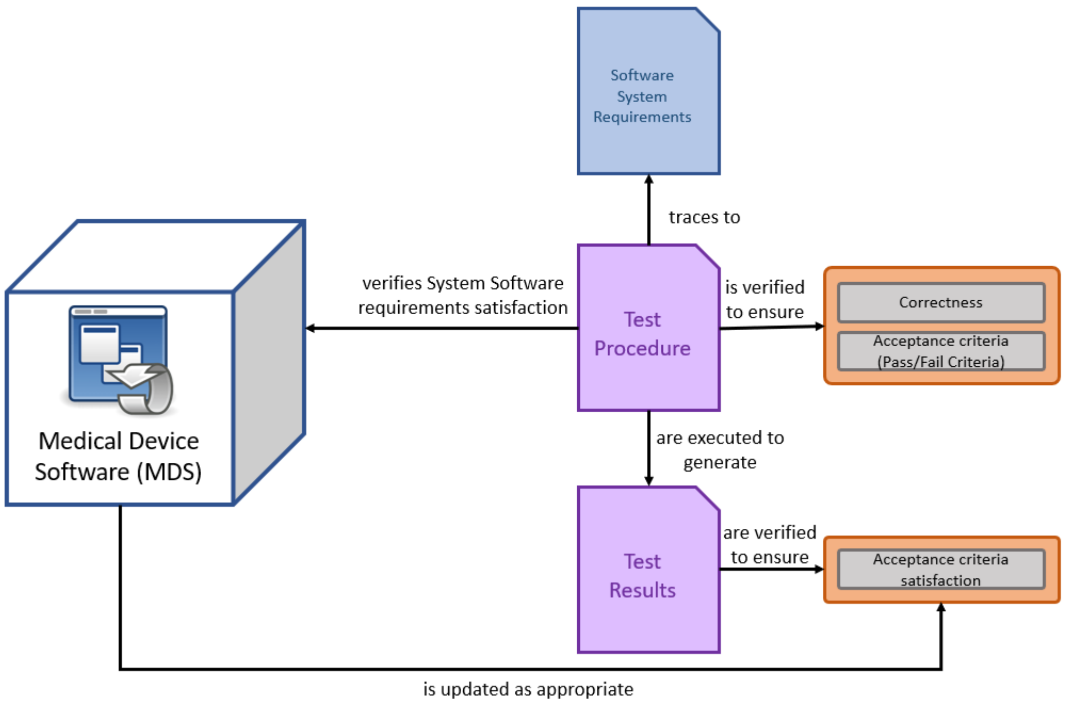

Software System Requirements
traces to
verifies System Software requirements satisfaction
Test Procedure
is verified to ensure
Correctness
Acceptance criteria (Pass/Fail Criteria)
Medical Device Software (MDS)
are executed to generate
Test Results
are verified to ensure
Acceptance criteria satisfaction
is updated as appropriate

Figura 3.11. Overview of the *Software system testing* activity

3. *Evaluate known residual anomalies* (clause 5.8.3);
4. *Document released versions* (clause 5.8.4);
5. *Document how released software was created* (clause 5.8.5);
6. *Ensure activities and tasks are complete* (clause 5.8.6);
7. *Archive software* (clause 5.8.7); and
8. *Assure reliable delivery of released software* (clause 5.8.8).

Table 3.11 presents the applicability of each task inside the software classes.

Tabela 3.11. Summary of tasks in *Software release* activity

Clause	Class A	Class B	Class C
5.8.1	X	X	X
5.8.2	X	X	X
5.8.3		X	X
5.8.4	X	X	X
5.8.5		X	X
5.8.6		X	X
5.8.7	X	X	X
5.8.8	X	X	X

As part of the task *Ensure software verification is complete*, the manufacturer shall ensure that all software verification activities have been completed and the results evaluated before the Software is released.

As part of the task *Document known residual anomalies*, the manufacturer shall document all known residual anomalies. Additionally, the manufacturer shall document all known residual anomalies to ensure that they do not contribute to an unacceptable risk as part of the task *Evaluate known residual anomalies*.

As part of the task *Document released versions*, the manufacturer shall document the version of the MDS that is being released. Furthermore, the manufacturer shall document the procedure and environment used to create the released software, as presented in task *Document how released Software was created*.

As part of the task *Ensure activities and tasks are complete*, the manufacturer shall ensure that all activities and tasks are complete along with all the associated documentation.

As part of the task *Archive software*, the manufacturer shall archive:

- The MDS and source-code; and
- The generated artifacts.

The archival is required for at least a period determined as the length of the device's lifetime as defined by the manufacturer or a time specified by relevant regulatory requirements.

As part of the task *Assure reliable delivery of released software*, the manufacturer shall establish procedures to ensure that the released MDS can be reliably delivered to the point of use without corruption or unauthorized change. These procedures shall address the production and handling of media containing the MDS, including as appropriate:

- Replication;
- Media labelling;
- Packaging;
- Protection;
- Storage; and
- Delivery.

Figure 3.12 presents an overview of the *Software release* activity.

3.6. Software Maintenance Process

The *Software maintenance process* contains 3 activities:

1. *Establish software maintenance plan*;

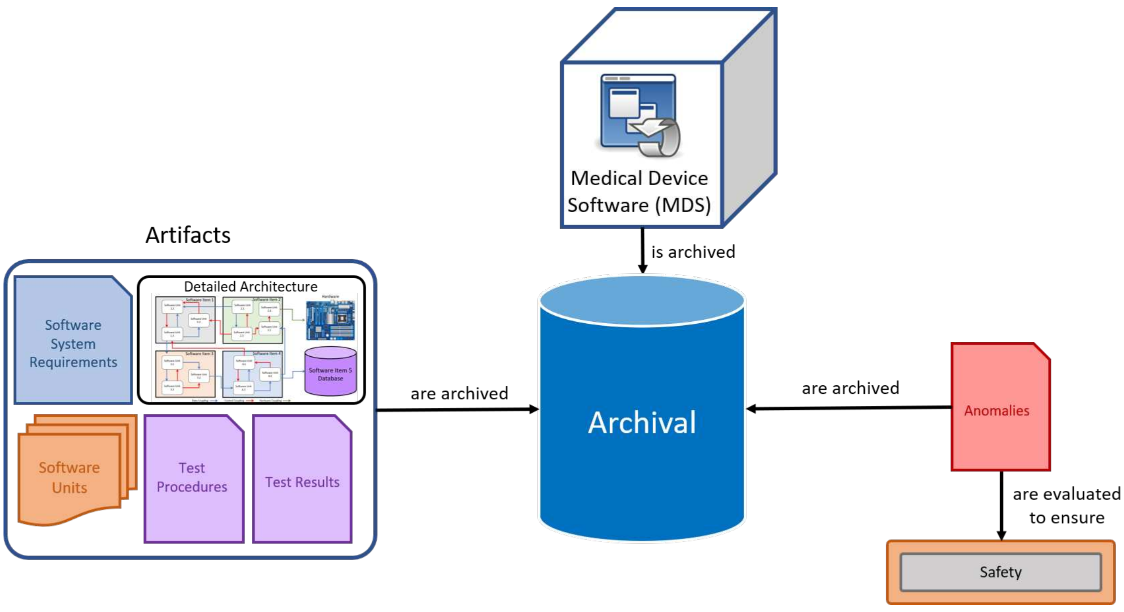


Figura 3.12. Overview of the *Software release* activity

2. *Problem and modification analysis*; and
3. *Modification implementation*.

3.6.1. Establish software maintenance plan

The Establish software maintenance plan (clause 6.1) contains 1 (one) task with the same name of the activity. As part of the task *Establish software maintenance plan*, the manufacturer shall establish a software maintenance plan (or plans) for conducting the activities and tasks of the maintenance process. The plan shall address the following:

- Procedures for feedback arising after the release of the MDS:
    1. Receiving;
    2. Documenting;
    3. Evaluating;
    4. Resolving; and
    5. Tracking.
- Criteria for determining whether the feedback is considered to be a problem;
- Use of the software problem resolution process for analyzing and resolving problems arising after the release of the MDS;
- Use of the software configuration management process for managing modifications to the existing software system; and
- Procedures to evaluate and implement:

1. Upgrades;
2. Bug fixes;
3. Patches; and
4. Obsolescence of SOUP.

Table 3.12 presents the applicability of each task inside the software classes.

Tabela 3.12. Summary of tasks in *Establish software maintenance plan* activity

Clause	Class A	Class B	Class C
6.1	X	X	X

3.6.2. Problem and modification analysis

The *Problem and modification analysis* activity contains 5 (five) tasks:

1. Document and evaluate feedback (clause 6.2.1);
2. Use software problem resolution process (clause 6.2.2);
3. Analyze change request (clause 6.2.3);
4. Change request approval (clause 6.2.4); and
5. Communicate to users and regulators (clause 6.2.5).

Table 3.13 presents the applicability of each task inside the software classes.

Tabela 3.13. Summary of tasks in *Problem and modification analysis* activity

Clause	Class A	Class B	Class C
6.2.1	X	X	X
6.2.2	X	X	X
6.2.3	X	X	X
6.2.4	X	X	X
6.2.5	X	X	X

The task *Document and evaluate feedback* is broken into 3 subtasks:

1. Monitor feedback (clause 6.2.1.1);
2. Document and evaluate feedback (clause 6.2.1.2); and
3. Evaluate problem report's affects on safety (clause 6.2.1.3).

As part of the subtask *Monitor feedback*, the manufacturer shall monitor feedback on MDS released for intended use.

As part of the subtask *Document and evaluate feedback*, the feedback shall be documented and evaluated to determine whether a problem exists in a released MDS. Any such problem shall be recorded as a Problem Report (PR). The PR shall include actual or potential adverse events and deviations from specifications and must be evaluated, as part of the subtask *Evaluate Problem Report's effects on safety*, to determine how it affects the safety of a released MDS and whether a change is needed to correct the problem.

As part of the task *Use software problem resolution process*, the manufacturer shall use the software problem resolution process to address PRs.

As part of the task *Analyze change requests*, the manufacturer shall analyze each Change Request (CR) for its effect on the organization, released MDS, and MS with which it interfaces.

As part of the task *Change request approval*, the manufacturer shall evaluate and approve CRs which modify released MDS. As required by local regulation, and presented by the task *Communicate to users and regulators*, the manufacturer shall inform users and regulators about:

- Any problem in released MDS and the consequences of continued unchanged use; and
- The nature of any available changes to released MDS and how to obtain and install the changes.

3.6.3. Modification implementation

The *Modification implementation* activity contains 2 (two) tasks:

1. *Use established process to implement modification* (clause 6.3.1); and
2. *Re-release modified MDS* (clause 6.3.2).

Table 3.14 presents the applicability of each task inside the software classes.

Tabela 3.14. Summary of tasks in *Modification implementation* activity

Clause	Class A	Class B	Class C
6.3.1	X	X	X
6.3.2	X	X	X

As part of the task *Use established process to implement modification*, the manufacturer shall use the software development process, briefly describe in Section 3.5 to implement the modifications needed by PRs or CRs. Modifications may be released as part of a complete re-release of a MDS or as a modification kit comprising changed SIs and the necessary tools to install the changes as modifications to an existing MDS, as presented in the task *Re-release modified MDS*.

3.7. Software Risk Process

The *Software risk process* contains 4 (four) activities:

1. *Analysis of software contributing to hazardous situations*;
2. *Risk control measures*;
3. *Verification of risk control measures*; and
4. *Risk management of software changes*.

3.7.1. Analysis of Software contributing to hazardous situations

The *Analysis of software contributing to hazardous situations* activity contains 4 (four) tasks:

1. *Identify software items that could contribute to a hazardous situation* (clause 7.1.1);
2. *Identify potential causes of contribution to a hazardous situation* (clause 7.1.2);
3. *Evaluate published SOUP anomaly lists* (clause 7.1.3); and
4. *Document potential causes* (clause 7.1.4).

Table 3.15 presents the applicability of each task inside the software classes.

Tabela 3.15. Summary of tasks in *Analysis of software contributing to hazardous situations* activity

Clause	Class A	Class B	Class C
7.1.1		X	X
7.1.2		X	X
7.1.3		X	X
7.1.4		X	X

As part of the task *Identify Software Items that could contribute to a hazardous situation*, the manufacturer shall identify SIs that could contribute to a hazardous situation identified in the Medical Device Risk Analysis of ISO 14971:2019 [ISO 2019]. The hazardous situation could be the direct result of software failure or failure of a risk control measure implemented in Software.

As part of the task *Identify potential causes of contribution to a hazardous situation*, the manufacturer shall identify potential causes of the SI identified above contributing to a hazardous situation. The manufacturer shall also consider potential causes including, as appropriate:

1. Incorrect or incomplete specification of functionality;
2. Software defects in the identified SI functionality;

3. Failure or unexpected results from SOUP;
4. Hardware failures or other software defects that could result in unpredictable software operation; and
5. Reasonably foreseeable misuse.

As part of the task *Evaluate published SOUP anomaly lists*, if a failure or unexpected results from SOUP is a potential cause of the SI contributing to a hazardous situation, the manufacturer shall evaluate as a minimum any anomaly list published by the supplier of the SOUP item relevant to the version of the SOUP item used in the MDS to determine if any of the known anomalies result in a sequence of events that could result in a hazardous situation.

As part of the task *Document potential causes*, the manufacturer shall document in the risk management file the potential causes of the SI contributing to a hazardous situation.

3.7.2. Risk control measures

The *Risk control measures* activity contains 2 (two) tasks:

1. *Define risk control measures* (clause 7.2.1); and
2. *Risk control measures implemented in software* (clause 7.2.2).

Table 3.16 presents the applicability of each task inside the software classes.

Tabela 3.16. Summary of tasks in *Risk control measures* activity

Clause	Class A	Class B	Class C
7.2.1		X	X
7.2.2		X	X

As part of the task *Define risk control measures*, for each potential cause of the software item contributing to a hazardous situation documented in the risk management file, the manufacturer shall define and document risk control measures. The risk control measures can be implemented in hardware, Software, working environment or user instruction.

As part of the task *Risk control measures implemented in software*, if a risk control measure is implemented as part of the functions of a SI, the manufacturer shall:

1. Include the risk control measure in the software requirements;
2. Assign a software safety class to the SI based on the possible effects of the hazard that the risk control measure is controlling; and
3. Develop the SI in accordance with the Software development process.

3.7.3. Verification of risk control measures

The *Verification of risk control measures* activity contains 2 (two) tasks:

1. *Verify risk control measures* (clause 7.3.1); and
2. *Document traceability* (clause 7.3.3).

Table 3.17 presents the applicability of each task inside the software classes.

Tabela 3.17. Summary of tasks in *Verification of risk control measures* activity

Clause	Class A	Class B	Class C
7.3.1		X	X
7.3.3		X	X

As part of the task *Verify risk control measures*, the implementation of each risk control measure documented shall be verified, and this verification shall be documented.

As part of the task *Document traceability*, the manufacturer shall document traceability of software hazards from:

1. The hazardous situation to the SI;
2. The SI to the specific software cause;
3. The software cause to the risk control measure; and
4. The risk control measure to its.

3.7.4. Risk management of software changes

The *Risk management of software changes* activity contains 3 (three) tasks:

1. *Analyze changes to Medical Device Software with respect to safety* (clause 7.4.1);
2. *Analyze impact of software changes on existing risk control measures* (clause 7.4.2); and
3. *Perform risk management activities based on analyses* (clause 7.4.3).

Table 3.18 presents the applicability of each task inside the software classes.

Tabela 3.18. Summary of tasks in *Risk management of software changes* activity

Clause	Class A	Class B	Class C
7.4.1	X	X	X
7.4.2		X	X
7.4.3		X	X

As part of the task *Analyze changes to Medical Device Software with respect to safety*, the manufacturer shall analyze changes to the MDS to determine whether:

1. Additional potential causes are introduced contributing to a hazardous situation; and
2. Additional software risk control measures are required.

As part of the task *Analyze the impact of software changes on existing risk control measures*, the manufacturer shall analyze changes to the Software, including changes to SOUP, to determine whether the software modification could interfere with existing risk control measures.

As part of the task *Perform risk management activities based on analyses*, the manufacturer shall perform all risk management activities defined in this section based on these analyses.

3.8. Software Configuration Management Process

The *Software configuration management process* contains 3 (three) activities:

1. *Configuration identification*;
2. *Change control*; and
3. *Configuration status accounting*.

3.8.1. Configuration identification

The *Configuration identification* activity contains 3 (three) tasks:

1. *Establish means to identify configuration items* (clause 8.1.1);
2. *Identify SOUP* (clause 8.1.2); and
3. *Identify system configuration documentation* (clause 8.1.3).

Table 3.19 presents the applicability of each task inside the software classes.

Tabela 3.19. Summary of tasks in *Configuration identification* activity

Clause	Class A	Class B	Class C
8.1.1	X	X	X
8.1.2	X	X	X
8.1.3	X	X	X

As part of the task *Establish means to identify configuration items*, the manufacturer shall establish a scheme for the unique identification of configuration items and their versions to be controlled for the project. This scheme shall include other MDS or entities such as SOUP and documentation.

As part of the task *Identify SOUP*, for each SOUP configuration item being used, including standard libraries, the manufacturer shall document:

1. The title,
2. The manufacturer, and
3. The unique SOUP designator.

As part of the task *Identify System configuration documentation*, the manufacturer shall document the set of configuration items and their versions that comprise the MDS configuration.

3.8.2. Change control

The *Change control* activity contains 3 (three) tasks:

1. *Approve change requests* (clause 8.2.1);
2. *Implement changes* (clause 8.2.2);
3. *Verify changes* (clause 8.2.3); and
4. *Provide means for traceability of change* (clause 8.2.4).

Table 3.20 presents the applicability of each task inside the software classes.

Tabela 3.20. Summary of tasks in *Change control* activity

Clause	Class A	Class B	Class C
8.2.1	X	X	X
8.2.2	X	X	X
8.2.3	X	X	X
8.2.4	X	X	X

As part of the task *Approve change requests*, the manufacturer shall provide configuration items only in response to an approved Change Request.

As part of the task *Implement changes*, the manufacturer shall implement the change as specified in the Change Request. In addition, the manufacturer shall identify and perform any activity that needs to be repeated as a result of the change, including changes to the software safety classification of MDS and SIs.

As part of the task *Verify changes*, the manufacturer shall verify the change, including repeating any verification that a change has invalidated.

As part of the task *Provide means for traceability of change*, the manufacturer shall create an audit trail whereby each of the following items are evaluated:

1. Change Request (CR);
2. Relevant Problem Report (PR); and
3. Approval of the CR.

3.8.3. Configuration status accounting

The *Configuration status accounting* (clause 8.3) contains 1 (one) task with the same name of the activity. As part of the task *Configuration status accounting*, the manufacturer shall retain retrievable records of the history of controlled configuration items, including System configuration. Table 3.21 presents the applicability of each task inside the software classes.

Tabela 3.21. Summary of tasks in *Configuration status accounting* activity

Clause	Class A	Class B	Class C
8.3	X	X	X

3.9. Software Problem Resolution Process

The *Software problem resolution* process contains 8 (eight) tasks:

1. *Prepare problem reports* (clause 9.1);
2. *Investigate the problem* (clause 9.2);
3. *Advise relevant parties* (clause 9.3);
4. *Use change control process* (clause 9.4);
5. *Maintain records* (clause 9.5);
6. *Analyze problems for trends* (clause 9.6);
7. *Verify software problem resolution* (clause 9.7); and
8. *Test documentation contents* (clause 9.8).

Table 3.22 presents the applicability of each task inside the software classes.

Tabela 3.22. Summary of tasks in *Software problem resolution* process

Clause	Class A	Class B	Class C
9.1	X	X	X
9.2	X	X	X
9.3	X	X	X
9.4	X	X	X
9.5	X	X	X
9.6	X	X	X
9.7	X	X	X
9.8	X	X	X

As part of the task *Prepare problem reports*, the manufacturer shall prepare a PR for each problem detected in a MDS. Problem Reports shall include a statement of criticality (for example, effect on performance, safety, or security) as well as other information that may aid in the resolution of the problem (for example, devices affected, supported accessories affected).

As part of the task *Investigate the problem*, the manufacturer shall:

1. Investigate the problem and, if possible, identify the causes;
2. Evaluate the problem's relevance to safety using the software risk management process;
3. Document the outcome of the investigation and evaluation; and
4. Create a Change Request(s) for actions needed to correct the problem or document the rationale for taking no action.

As part of the task *Advise relevant parties*, the manufacturer shall advise relevant parties of the existence of the problem, as appropriate.

As part of the task *Use change control process*, the manufacturer shall approve and implement all Change Requests, observing the requirements of the Change control process.

As part of the task *Maintain records*, the manufacturer shall maintain records of PRs and their resolution, including their verification. Additionally, the manufacturer shall perform analysis to detect trends in Problem Reports.

As part of the task *Verify software problem resolution*, the manufacturer shall verify resolutions to determine whether:

1. Problem Report has been resolved, and the Problem Report has been closed;
2. Adverse trends have been reversed;
3. Change Requests have been implemented in the appropriate MDS and activities; and
4. Additional problems have been introduced.

As part of the task *Test documentation contents*, when testing, retesting or regression testing SIs and MDS following a change, the manufacturer shall include in the test documentation:

1. Test results;
2. Anomalies found;
3. The version of Software tested;
4. Relevant hardware and software test configurations;
5. Relevant test tools;
6. Date tested; and
7. Identification of the tester.

3.10. Relationship with other Standards

As described before, the IEC 62304 applies to the development and maintenance of MDS. The Software is considered a part of the MD. The IEC 62304 should be used together with other appropriate standards when developing an MD. Medical device management standards such as ISO 13485:2016 [ISO 2016a] and ISO 14971:2019 [ISO 2019] provide a management environment that lays a foundation for an organization to develop products. Safety standards such as IEC 60601-1-12:2014/AMD 1:2020 [IEC 2020], IEC 61010-1:2010 [IEC 2010a], and IEC 82304-1:2016 [IEC 2016] give specific direction for creating safe Medical Devices. When Software is a part of these Medical Devices, IEC 62304 provides more detailed direction on developing and maintaining safe MDS. Many other standards such as ISO/IEC/IEEE 12207:2017 [ISO 2017], IEC 61508-3:2010 [IEC 2010b], and ISO/IEC/IEEE 90003:2018 [ISO 2018] can be looked to as a source of methods, tools and techniques that can be used to implement the requirements in IEC 62304.

Figure 3.19 shows the relationship among these standards.

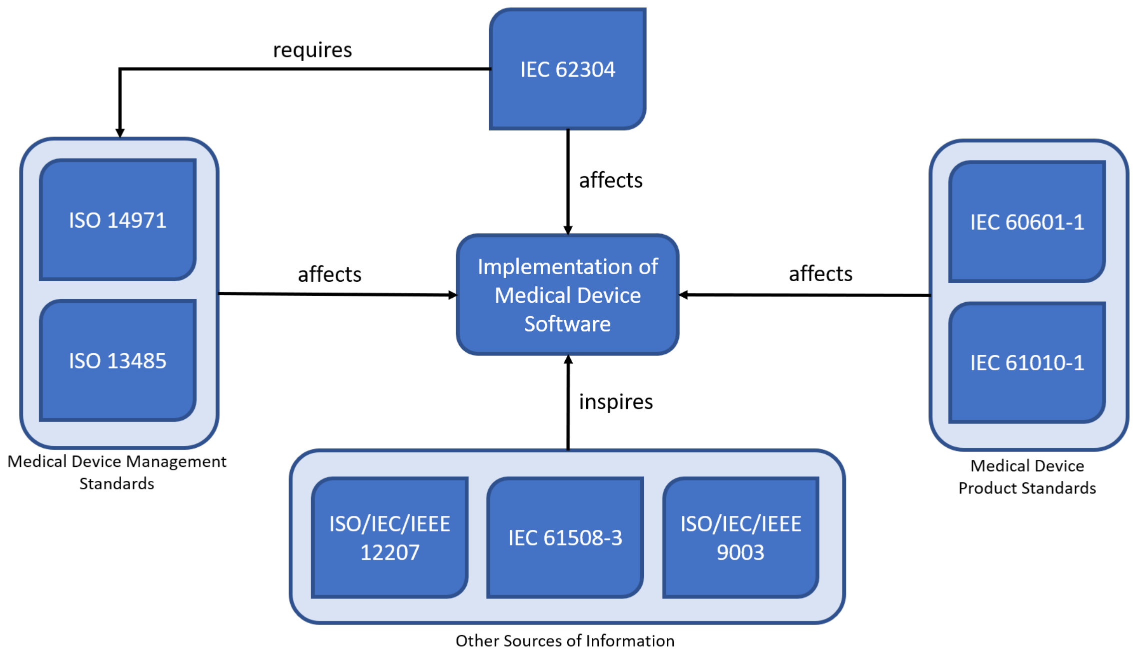

requires
IEC 62304
affects
ISO 14971
ISO 13485
Medical Device Management Standards
affects
Implementation of Medical Device Software
affects
IEC 60601-1
IEC 61010-1
Medical Device Product Standards
inspires
ISO/IEC/IEEE 12207
IEC 61508-3
ISO/IEC/IEEE 9003
Other Sources of Information

Figura 3.13. Relationship with other standards

3.11. Agile Methods and Scrum

According to Davis (2013), in 2001, a group of 17 professionals met and produced a document that established the principles of agile development. This document, called the Manifesto for Agile Development (MAD), was prepared broadly and generically. These general terms and concepts started to guide the agile way of managing projects.

Several proponents of agile methods agreed with the MDA [Beck et al. 2001], shown in Figure 3.14. It synthesizes the origins of a set of lightweight methodologies, such as Scrum [Schwaber and Beedle 2001], Extreme Programming (XP) [Beck 2000], Crystal [Cockburn 2004], Feature Driven Development (FDD) [Palmer and Felsing 2002], Test Driven Development (TDD) [Astels 2003], Dynamic Software Development Method (DSDM) [Stapleton 1997] and Adaptive Software Development (ASD) [Highsmith 2000].

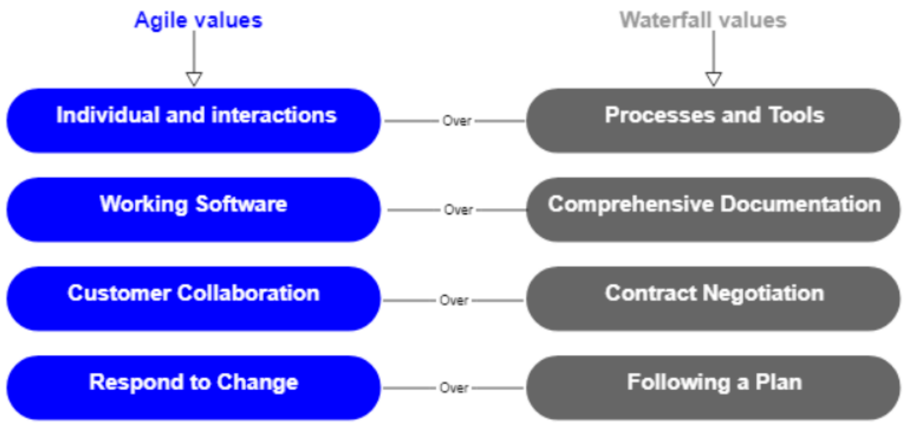


Figura 3.14. Manifesto for Agile Development (MAD) [Beck et al. 2001]

In addition to these fundamental values identified, the participants of the MAD also created 12 principles that guide the agile development of Software [Davis 2013]:

1. Our highest priority is to satisfy the customer through the early and continuous delivery of valuable Software;
2. Accept changing requirements, even at the end of development. Agile processes are adapted to changes so that the client can gain competitive advantages;
3. Deliver Software frequently running, on the scale of weeks to months, with a preference for shorter periods;
4. Business-related people and developers must work together daily throughout the project;
5. Build projects around motivated individuals. Giving them the necessary environment and support and trusting that they will do their job;
6. The most efficient and effective method of transmitting information to a development team is through face-to-face conversation;
7. Functional Software delivery is the primary measure of progress;
8. Agile processes promote a sustainable environment. Sponsors, developers and users must be able to maintain constant steps indefinitely;
9. Continuous attention to technical excellence and good design increases agility;
10. Simplicity;
11. The best architectures, requirements and designs emerge from self-organizing teams; and

12. At regular intervals, the team reflects on becoming more effective, so it adjusts and optimizes its behavior accordingly.

Principles 1, 2 and 7 are strongly correlated. In principle 2, breaking the paradigm between agile and traditional development stands out since traditional development avoids and makes it challenging to change requirements. In traditional projects, it is essential to follow a plan, and any variation can mean a significant risk to the project's success. In agile projects, on the other hand, the customer is the one who dictates the priorities, and if the changes add more value to the customer, these are welcome, as stated in principle 1. In principle 7, as functional Software is the primary measure of progress, the added value to the customer is the same as the working Software.

In principle 3, agile developments work with the concept of iterations, with cyclical efforts of fixed duration. The work to be done is selected and prioritized before the start and then delivered at the end. This principle strongly correlates with 9, providing a well-designed software with incremental delivery and modularity.

Principles 4, 5, 6 and 11 are associated with human relationships. Principle 4 focuses on free and unhindered communication from any barrier. Principles 5 and 11 provide that teams are self-organizing and do not need someone by their side to tell them what to do. Principle 6 establishes a preference for verbal and informal communication overwritten and formal communication.

The rationale for principle 8 is that projects can keep pace indefinitely without the team experiencing fatigue. Principle 10 focuses on keeping things simple and eliminating what is considered unnecessary. Finally, principle 12 appears in several methods with "Retrospective" calls. At the end of each iteration, the team looks at the completed work and reflects on what went right and what went wrong.

According to Stober and Hansmann (2010), agile thinking is an attempt to simplify things, reducing planning complexity, focusing on customer value and shaping a favorable climate for participation and collaboration. Furthermore, Vuori(2011) points out that there is a tendency in companies to transform their Software practices and product development to a more incremental way, using agile development.

Sutherland (2010) defined Scrum as an iterative and incremental framework for application development, and its structure is defined in work cycles, which happen as a race, called Sprint. One Sprint typically has 1 (one) to 4 (four) weeks, ending on a specific date, regardless of the work being completed, and is never extended. Scrum has a series of defined roles with different responsibilities, as shown in Table 3.23.

The Scrum Master protects the team by ensuring that it does not over-commit itself to what it can accomplish during a sprint. He also acts as a facilitator and becomes responsible for removing any obstacles that Scrum Team raises during these meetings.

A meeting called Sprint Planning takes place at the beginning of each Sprint. The Product Owner is a role of important responsibility and visibility in the Scrum method. This represents the customer in decisions and prioritization, considering the value added to the product.

The Product Owner and Scrum team review the product backlog and discuss the

Tabela 3.23. Roles of Scrum

Role	Description
Product Owner	Defines the items that make up the product backlog and prioritizes them in sprint planning.
Scrum Master	Ensures that the team respects and follows Scrum values and practices. It also protects the team by ensuring that it does not overcommit itself to what it is capable of accomplishing during a sprint.
Scrum Team	Formed by the development team. There is not necessarily a functional division through traditional roles such as programmer, test analyst or architect. Everyone on the project works together to complete the set of work they have jointly committed to for a sprint.

goals and context for the items. In addition, the Scrum Team selects the items from the product backlog and commits to completing, by the end of the Sprint, forming the sprint backlog.

A Release is the delivery of one or more product increments, generated in one or more successive sprints, for use. Scrum projects perform frequent releases. Performing releases throughout a project gets frequent feedbacks and promote a sense of development evolution.

The Product Backlog is a list containing all the desired functionality for a product. The Product Owner defines the content of this list. The Product Backlog does not need to be complete at the beginning of a project. Instead, it can start with whatever is most apparent at first. Over time, the Product Backlog grows and changes as the team learns more about the product and its users.

The Sprint Backlog is a list of tasks that the Scrum Team undertakes to do in a Sprint. The Sprint backlog items are extracted from the Product Backlog by the Scrum Team based on the priorities set by the Product Owner and the team's perception of the time needed to complete the various functionality.

After the Sprint is completed, there is a Sprint Retrospective, where the team and stakeholders discuss and review the results, identifying applicable improvements. Figure 3.15 presents the Scrum structure.

In a Scrum project, the team monitors its progress against a plan, updating a Burndown Chart at the end of each Sprint. The vertical axis of a Burndown Chart shows the amount of work that still needs to be done at the beginning of each Sprint. The horizontal axis represents the measure of time. Figure 3.16 presents an example of the Burndown Chart.

3.12. Agile Software Development Model

The Agile Software Development Model presented in Figure 3.17 consists of 7 stages. Stage A is planning, where the two plans required for IEC 62304 are generated. These plans should describe the other stages of the agile software development model proposed in this work, including the roles, responsibilities and tools used in software development.

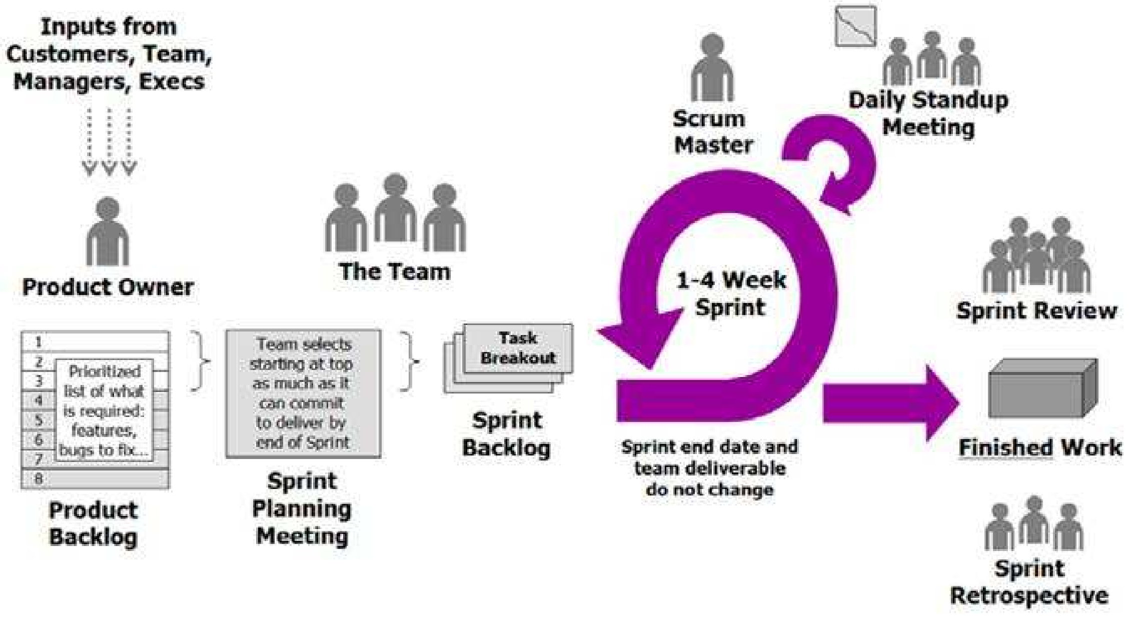

Inputs from Customers, Team, Managers, Execs
Product Owner
The Team
Scrum Master
Daily Standup Meeting
1-4 Week Sprint
Sprint Review
Prioritized list of what is required: features, bugs to fix...
Team selects starting at top as much as it can commit to deliver by end of Sprint
Task Breakout
Sprint Backlog
Sprint end date and team deliverable do not change
Finished Work
Product Backlog
Sprint Planning Meeting
Sprint Retrospective

Figura 3.15. Scrum framework [Beck et al. 2001]

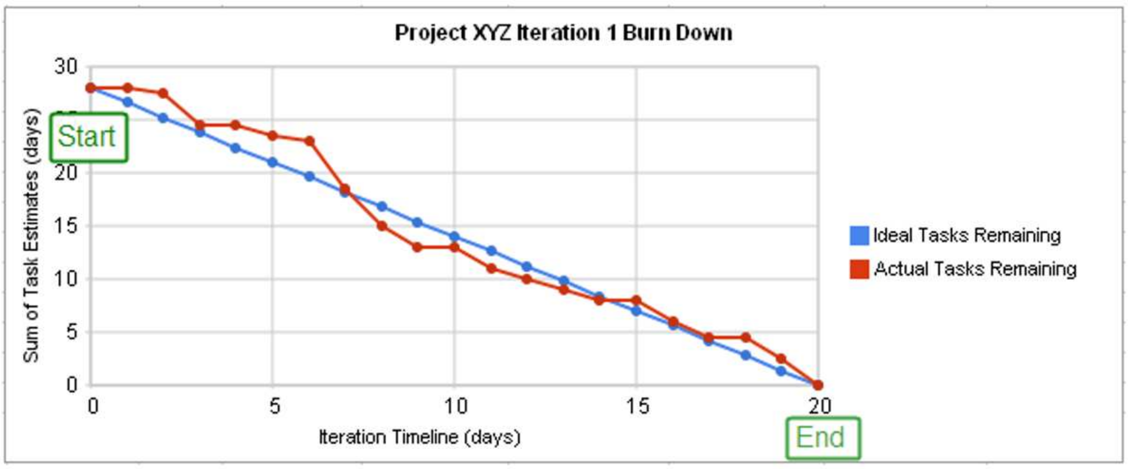


Figura 3.16. Example of Burn Down Chart [Ambler 2002]

Stage B represents the Architecture Design, involving the software items and accommodating the future allocation of Software System requirements. Stage A and B represent the inputs needed for the repetitive execution of sprints.

Stages C and D are within the Sprint. Stage C represents the Sprint Planning with a focus on prioritization of the system requirements. In stage D, the specification of Software Requirements is from the refinement of the prioritized System Requirements. With the Software Requirements and the Architecture, a Detailed Architecture is generated specifying the software units for this Sprint's software requirements implementation, thus generating an implementation of each software unit. Also, within the sprints, a set of testing procedures for the software units is built and executed, thus causing the test results at the unit level. Finally, Sprints are repeated until all software units are architecturally detailed, implemented and tested.

Once the implementations of the software units are completed, stage E is the integration implementation and testing. At this stage, the Software Units built within the Sprints will be integrated into Software Items. In this same stage, the data and control couplings between the Software Units belonging to the same software item will be evaluated, respecting the detailed specification of the architecture. In stage F, the system testing will be done, involving integrating the Software Items and the hardware designated for the MDS. Finally, in stage G, the MDS will be released for its use.

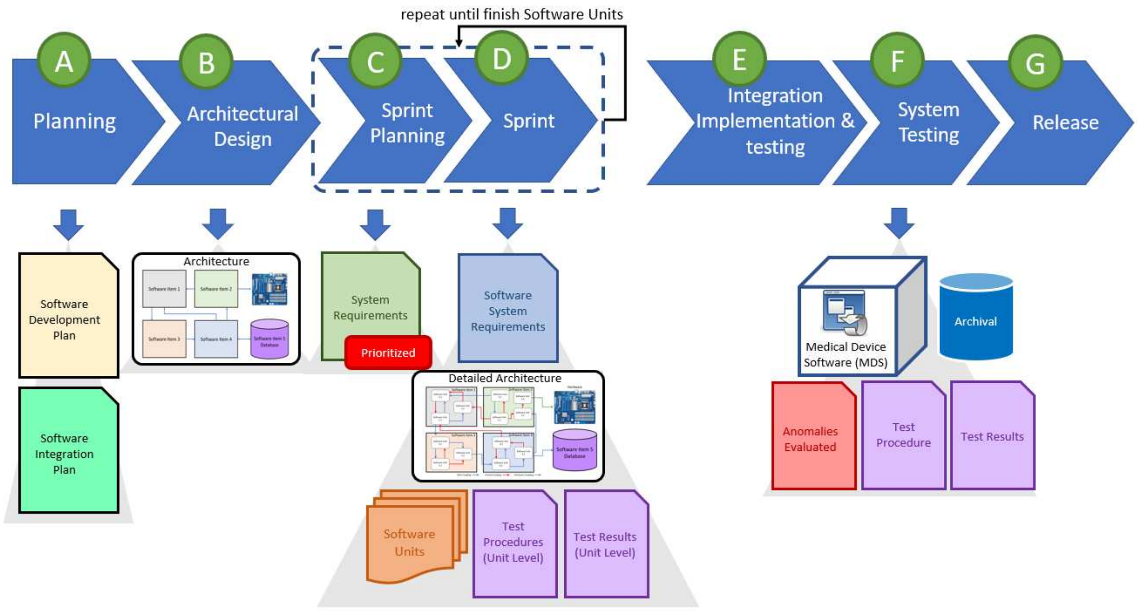

repeat until finish Software Units
A
Planning
B
Architectural Design
C
Sprint Planning
D
Sprint
E
Integration Implementation & testing
F
System Testing
G
Release
Software Development Plan
Software Integration Plan
Architecture
System Requirements
Prioritized
Software System Requirements
Detailed Architecture
Software Units
Test Procedures (Unit Level)
Test Results (Unit Level)
Medical Device Software (MDS)
Archival
Anomalies Evaluated
Test Procedure
Test Results

Figura 3.17. Agile software development model

During the sprint planning meeting (Figure 3.18 - stage C), the system requirements are evaluated. The team chooses those system requirements that will be implemented by Software, which are prioritized into the Sprint. Once the System Requirements set is selected, the scrum master monitors the progress of its refinements in software requirements and other development process artifacts, using the Burn Down Chart. During the Sprint (Figure 3.18 - stage D), which can last up to 4 weeks, the development of the Software System requirements, the detailed Architecture, implementation of the Software Units, and their Test procedures and Test Results are carried out in this time. The Weekly Meeting evaluates the progress of refining the system requirements prioritized for this Sprint. The Sprint Review ensures that generated artifacts have been verified at the Software Unit level. The team will also carry out a Sprint Retrospective that will record and evaluate the possible Anomalies identified in the artifacts generated in this Sprint, scheduling the correction of these anomalies for a future Sprint. Stages C and D are detailed in Figure 3.18.

According to Figure 3.19, after all the software units are developed within the numerous sprints performed, the integration implementation Testing, stage E, performs the verification of the integration between the software units belonging to the same software item. Therefore, test procedures at the integration level will be created and generated

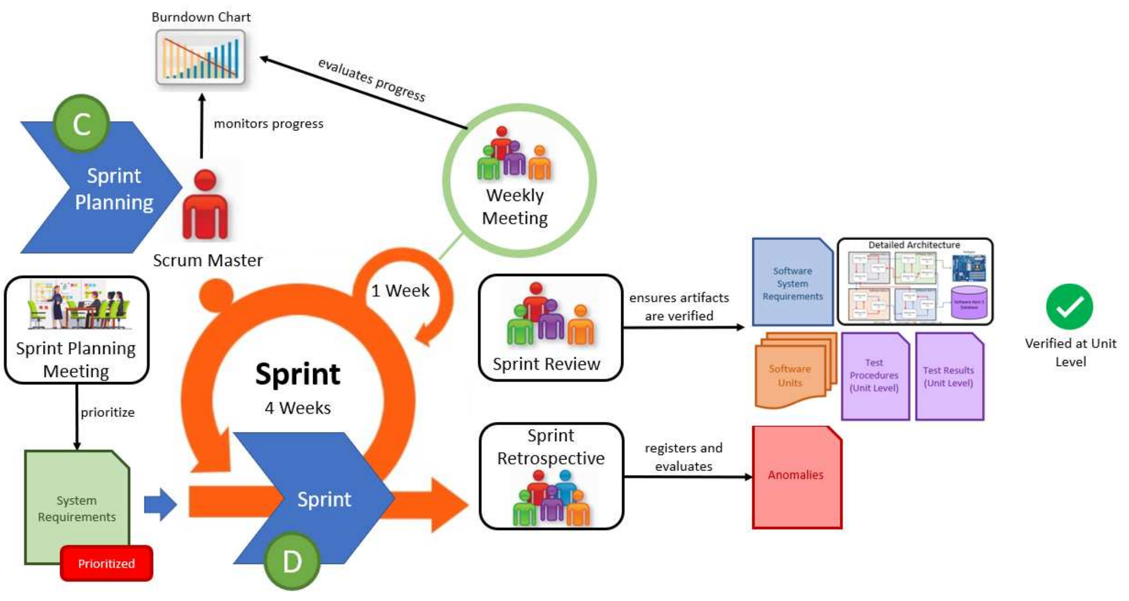


Figura 3.18. Sprint structure

from these tests when executed. In system testing, stage F, on the other hand, performs the verification of the integration between the software items with the definition of test procedures that exercise the software requirements and that, when executed, generate test results. Finally, in stage G, the manufacturer archives the artifacts generated during the Medical device software and evaluate the possible anomalies identified regarding the safety aspect.

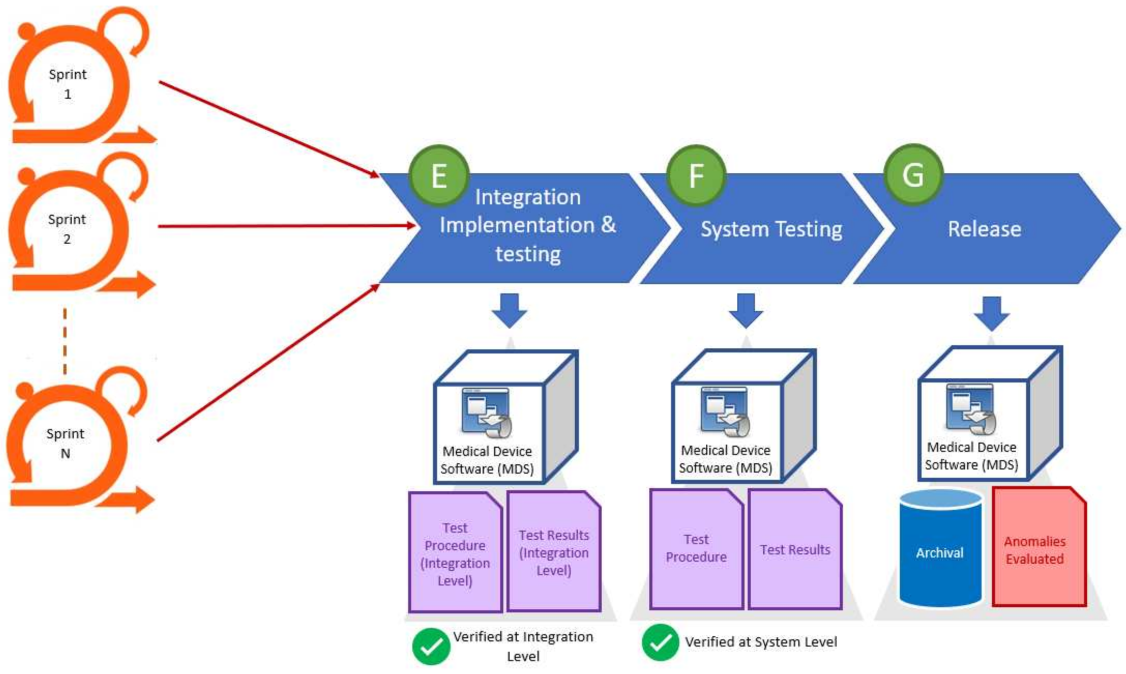

Sprint 1
Sprint 2
Sprint N
E
Integration Implementation & testing
F
System Testing
G
Release
Medical Device Software (MDS)
Test Procedure (Integration Level)
Test Results (Integration Level)
Verified at Integration Level
Medical Device Software (MDS)
Test Procedure
Test Results
Verified at System Level
Medical Device Software (MDS)
Archival
Anomalies Evaluated

Figura 3.19. Integration, system testing, and release

3.13. Related Work

For the identification of the related work, we executed a Systematic Literature Mapping (SLM) that was published in March 2021 in Journal of Health Informatics [Marques et al. 2021].

3.13.1. Systematic Literature Mapping (SLM)

Our previous work [Marques et al. 2021] has identified works reporting the usage of IEC 62304. We also identified the advantages and difficulties of using IEC 62304. We found and classified 22 (twenty-two) works that met the inclusion criteria, as part of our SLM. Using the instructions suggested by Petersen et al. (2015), 4 Categories (Cat) were defined and identified (Cat1...4):

- Conceptual Analysis (Cat1): works that discuss a theoretical concept or a new approach, but without validating it;
- Experimental Analysis (Cat2): works that discuss a theoretical concept or a new approach with validation;
- Experience Report (Cat3): works that report an industrial experience without declaring research questions or theoretical concepts; and
- Survey (Cat4): works that collect data based on a questionnaire.

The IEC 62304 standard presents software development and support processes such as configuration and risk control. Thus, the works usually present contributions that describe, support, elucidate their processes and activities through guides, models, methods or comparatives. Thus, the Contribution (Co) axis identified 4 (four) types of contribution (Co1...4), as follows:

- Guidance (Co1): works that support the understanding of IEC 62304 processes and activities;
- Model (Co2): works that present an extension and detailing of the processes and activities of IEC 62304, with reusable tools, methods and checklists;
- Method (Co3): works that present methods to meet only one IEC 62304 process or activity; and
- Comparative (Co4): works that present a comparison of IEC 62304 in some perspective.

IEC 62304 describes 5 (five) processes: Software Development Process, Software Maintenance Process, Software Risk Management Process, Software Configuration Management Process, and Software Problem Resolution Process. The Software Development Process contains 8 (eight) activities, as presented in sections 3.5. For the Variability (Var) axis, the team leading the SLM decided to group 4 (four) processes and 8 (eight) activities of the Software Development Process into 4 (four) groups with identification (Var1...4). We did not find any work associated with the Software Maintenance Process:

- Planning and Requirements (Var1): works that address the Development Planning and Requirements Analysis activities that belong to the Software Development Process;
- Design and Implementation (Var2): works that address the Architectural Design, Detailed Design and Unit Implementation and Verification activities that also belong to the Software Development Process;
- Tests (Var3): works that address the Integration and Integration Testing and Software System Testing activities that belong to the Software Development Process; and
- Risks (Var4): works that address the Software Risk Management process.

3.13.2. Results

Figure 3.20 presents the 22 works grouped into the 3 (three) axis. We mapped some works to more than one possibility within the same axis.

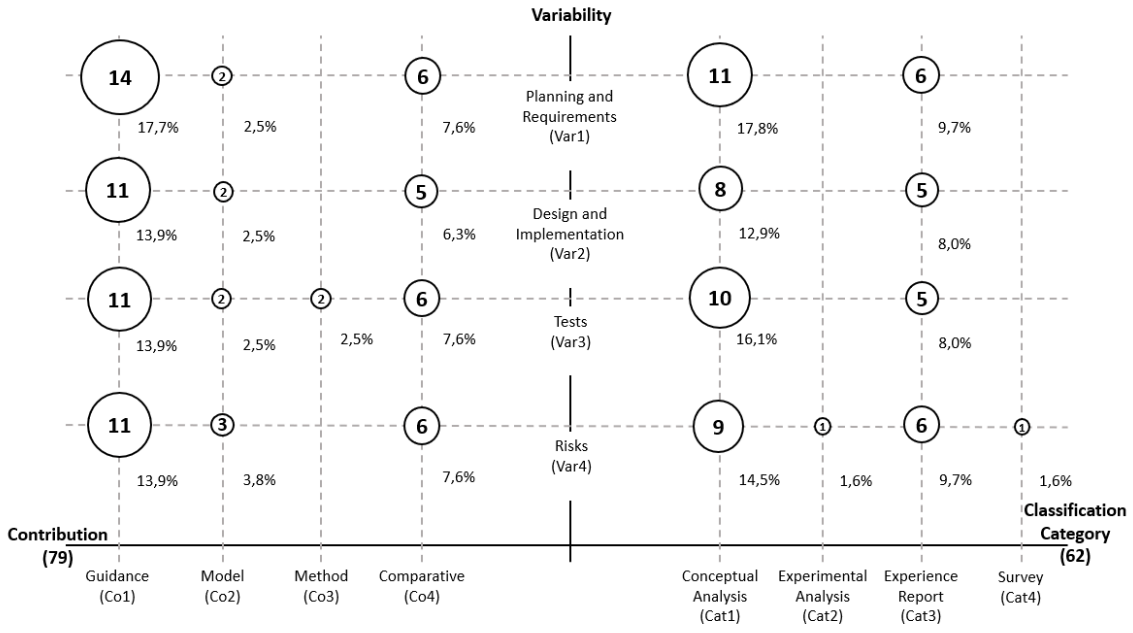


Figura 3.20. SLM bubble chart

Jordan (2006) and Varri et al. (2019) described IEC 62304 presenting their software processes and classes. Jordan (2006) was the first to deal with IEC 62304 after its issuance. Varri et al. (2019) plays a similar role but updated with the IEC 62304 amendment issued in 2015.

Huhn and Zechner (2010) proposed a method to evaluate arguments centred on quality and engineering in reliability cases (assurance cases) to guarantee software development, according to IEC 62304.

Mc Caffery et al. (2010) compared the depth of current medical equipment regulations, focusing on IEC 62304, concerning Capability Maturity Model Integration (CMMI) in specifying which risk management practices companies should adopt when developing software medical devices. Bianco (2011) describes a quality system that integrates as main processes those specified by IEC 62304 and applies a risk-oriented approach and supporting processes such as contract and supplier management.

Cruciani and Vicario (2011) identified the need for the extensive testing effort necessary to comply with IEC 62304 prescriptions, presenting how data flow analysis can identify an appropriate set of constraints explored in the verification stage at reducing the set of tests, preserving the coverage. Finally, Larson et al. (2012) presented an initial proposal for a real-time and critical computing platform to integrate heterogeneous devices, identifying the absence of requirements in IEC 62304 for this purpose.

McHugh et al. (2012) identified how regulations affect medical device software development companies, and they made recommendations on compliance with IEC 62304. Wong and Callaghan (2012) described an approach taken to manage the baselines of software requirements for medical devices in need of compliance with IEC 62304.

Regan et al. (2013) described the extent and diversity of traceability requirements in medical device standards and guidelines at each stage of the software development life cycle, as required by IEC 62304.

Höss et al. (2014) described the first experiences with the implementation of IEC 62304 to guarantee the quality of a radiotherapy unit, being the only work with an industry report.

Rust et al. (2016) described a roadmap that assists small and medium-sized companies in developing medical Software by IEC 62304, offering design patterns to generate pre-established artifacts and models to demonstrate compliance. They also presented a software development plan to help organizations in which the use of IEC 62304 can be problematic because they are new organizations or have limited experience in the medical field. They also present a roadmap, divided into two levels: (a) the high level consists of the activities and tasks necessary for implementing IEC 62304, and (b) the low level contains the artifacts of design standards and instructions related to the tasks. The script involved a consultation, by questionnaire, with 6 (six) experts performing the evaluation.

Laukkarinen et al. (2017) examined the obstacles and benefits of using DevOps to develop Software for medical devices. Finally, Hatcliff provided an overview of the life cycle problems of interoperable medical devices not sufficiently addressed in existing medical device standards, including IEC 62304.

Kasisopha and Meananeatra (2019) presented a directive for Very Small Entities (VSE) that employ ISO TR 29110 [ISO 2016b] and aspire to apply the IEC 62304 processes in constructing Software for medical devices. Marques and Yelisetty (2019) analyzed the characteristics of specification of software requirements in regulated environments such as medical, aeronautical and rail. The four characteristics identified are consistency (internal and external), unambiguity, verifiability and traceability. The document also describes the three standards used in these regulated environments (RTCA DO-178C, IEC 62279 and IEC 62304). It examines their similarities and differences from the point of view of the requirements' specification.

Table 3.24 identified the difficulties inside the 22 works.

Tabela 3.24. Difficulties of using IEC 62304 [Marques et al. 2021]

Difficult	Description
DIF 1	Challenging to select and use automated tools to comply with the standard.
DIF 2	High initial effort with a learning curve to overcome.
DIF 3	There is a lack of recommendations of methods and techniques.
DIF 4	High test effort.
DIF 5	Difficult to use in small and medium-sized companies.
DIF 6	Lack of how to handle the interoperability of various medical products.
DIF 7	Continuous integration is complex.
DIF 8	Lack of integration with CMMI [Chrissis et al. 2011].
DIF 9	Need for qualified personnel.

Table 3.25 identified the advantages inside the 22 works.

Tabela 3.25. Advantages of using IEC 62304 [Marques et al. 2021]

Advantage	Description
ADV 1	Present rigorous criteria equivalent to the norms of other critical domains.
ADV 2	Do not prescribe a specific lifecycle, only its processes.
ADV 3	Facilitate competitiveness among companies.
ADV 4	Determine rigor according to safety impact.
ADV 5	Control of software planning, programming, testing and documentation.
ADV 6	Present a process for handling software updates.
ADV 7	Emphasize the importance of requirements management and traceability.

3.14. Final Considerations

This chapter presented the fundamentals of IEC 62304 with an Agile Software Development Model. In 2021, IEC 62304 completed 15 years. Thus, the authors believe that there are already works that report experiences, analyzes and difficulties in its use. These results can be interesting to direct further research and definitions of methods, models, guides or other materials to comply with IEC 62304.

The main contributions of this chapter are:

1. The summary of IEC 62304 processes, activities, and tasks, as presented in Sections 3.5, 3.6, 3.7, 3.8, and 3.9;

2. The illustration of Figures 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, and 14 that contributes to a visual understanding of the Software Development Process;

3. The Agile Software Development Model providing an adaptation of Scrum, focusing on IEC 62304 compliance; and

4. The summary of the Systematic Literature Mapping performed.

We believe that our chapter helps the difficulty identified during SLM and presented in Table 3.24. Furthermore, by creating a lecture involving IEC 62304, we are helping to solve DIF 2 *High initial effort with a learning curve to overcome* and DIF 9 *Need for qualified personnel*, because this chapter allows readers to better understand with a qualification in the IEC 62304.

We also believe that our Agile Software Development Model is helpful to solve some difficulties identified during SLM, as presented in Table 3.24. We are helping to solve DIF 3 *There is a lack of recommendations of methods and techniques* and DIF 5 *Difficult to use in small and medium-sized companies*. The usage of an adaptation of Scrum, which focuses on small and medium teams (4 to 9 participants), we adapted the compliance using the Agile Software Development Model presented in Section 3.12, facilitating the competitiveness among companies. We identified that our Agile Software Development Model reaffirms the ADV 2 *Do not prescribe a specific lifecycle, only its processes* and ADV 3 *Facilitate competitiveness among companies*.

Referências

[Ambler 2002] Ambler, S. (2002). *Agile Modeling*. Wiley, Nova Iorque, Estados Unidos.

[Astels 2003] Astels, D. (2003). *Test-driven Development: A Practical Guide*. Pearson Education.

[Beck 2000] Beck, K. (2000). *Extreme Programming Explained: Embrace Change*. Addison-Wesley.

[Beck et al. 2001] Beck, K., Fulano, Beltrano, and Ciclano (2001). Manifesto for agile software development.

[Bianco 2011] Bianco, C. (2011). Integrating a risk-based approach and iso 62304 into a quality system for medical devices. In *Nineteenth Safety-Critical Systems Symposium*.

[Caffery et al. 2010] Caffery, F. M., Burton, J., and Richardson, I. (2010). Risk management capability model for the development of medical device software. *Software Quality Journal*, 18(1):81–107.

[Chrissis et al. 2011] Chrissis, M. B., Konrad, M., and Shrum, S. (2011). *CMMI for Development: Guidelines for Process Integration and Product Improvement*. Software Engineering Institute.

[Cockburn 2004] Cockburn, A. (2004). *Crystal Clear: a Human-powered Methodology for Small Teams*. Addison-Wesley.

[Cruciani and Vicario 2011] Cruciani, F. and Vicario, E. (2011). Reducing complexity of data flow testing in the verification of a iec-62304 flexible workflow system. In *30th International Conference (SAFECOMP)*.

[Davis 2013] Davis, B. (2013). *Agile Practices for Waterfall Projects*. J.ROSS.

[Highsmith 2000] Highsmith, J. A. (2000). *Adaptive Software Development: a Collaborative Approach to Managing Complex Systems*. Dorset House Publishing.

[Huhn and Zechner 2010] Huhn, M. and Zechner, A. (2010). Arguing for software quality in an iec 62304 compliant development process. In *4th International Symposium on Leveraging Applications*.

[Höss et al. 2014] Höss, A., Lampe, C., Panse, R., Ackermann, B., Naumann, J., and Jäkel, O. (2014). First experiences with the implementation of the european standard en 62304 on medical device software for the quality assurance of a radiotherapy unit. *Radiat Oncol*, 9(79):1–10.

[IEC 2006] IEC (2006). Iec 62304:2006 medical device software - software life-cycle processes – amendment 1. Technical report, International Electrotechnical Commission.

[IEC 2010a] IEC (2010a). Iec 61010-1:2010 safety requirements for electrical equipment for measurement, control, and laboratory use - part 1: General requirementssafety requirements for electrical equipment for measurement, control, and laboratory use - part 1: General requirements. Technical report, International Electrotechnical Commission.

[IEC 2010b] IEC (2010b). Iec61508-3:2010 functional safety of electrical/electronic/programmable electronic safety related sysyste - software requirements.

[IEC 2015] IEC (2015). Iec 62304:2006/amd 1:2015 medical device software - software life-cycle processes – amendment 1. Technical report, International Electrotechnical Commission.

[IEC 2016] IEC (2016). Iec 82304-1:2016 health software - part 1: General requirements for product safety. Technical report, International Electrotechnical Commission.

[IEC 2020] IEC (2020). Iec 60601-1-12:2014/amd 1:2020 medical electrical equipment part 1-12: General requirements for basic safety and essential performance — collateral standard: Requirements for medical electrical equipment and medical electrical systems intended for use in the emergency medical services environment. Technical report, International Electrotechnical Commission.

[ISO 2016a] ISO (2016a). Iso 13485:2016 medical devices — quality management systems — requirements for regulatory purposes. Technical report, International Standardization.

[ISO 2016b] ISO (2016b). Iso tr 29110:2016 systems and software engineering — lifecycle profiles for very small entities (vses). Technical report, International Standardization Organization.

[ISO 2017] ISO (2017). Iso/iec/ieee 12207:2017 systems and software engineering — software life cycle processes. Technical report, Internation Standardization Organization.

[ISO 2018] ISO (2018). Iso/iec/ieee 90003:2018 software engineering — guidelines for the application of iso 9001:2008 to computer software. Technical report, International Standardization.

[ISO 2019] ISO (2019). Iso 14971:2019 medical devices — application of risk management to medical devices. Technical report, International Standardization.

[Jordan 2006] Jordan, P. (2006). Standard iec 62304 - medical device software - software lifecycle processes. In *IET Seminar on Software for Medical devices*.

[Kasisopha and Meananeatra 2019] Kasisopha, N. and Meananeatra, P. (2019). Applying iso/iec 29110 to iso/iec 62304 for medical device software sme. In *2nd International Conference on Computing and Big Data*.

[Larson et al. 2012] Larson, B., Hatcliff, J., Procter, S., and Chalin, P. (2012). Requirements specification for apps in medical application platforms. In *4th International Workshop on Software Engineering in Health Care (SEHC)*.

[Laukkarinen et al. 2017] Laukkarinen, T., Kuusinen, K., and Mikkonen, T. (2017). Devops in regulated software development: Case medical devices. In *2017 IEEE/ACM 39th International Conference on Software Engineering: New Ideas and Emerging Technologies Results Track (ICSE-NIER)*.

[Magnuson 2012] Magnuson, A. (2012). Iec/iso 62304 regulations for the development of medical software devices. Master's thesis, Chalmers University of Technology.

[Marques 2019] Marques, J. (2019). Uma análise das características de especificação de requisitos de software em normas de ambientes regulados. In *22º Workshop de Engenharia de Requisitos (WER 2019)*.

[Marques and Cunha 2019] Marques, J. and Cunha, A. (2019). Ares: An agile requirements specification process for regulated environments. *International Journal of Software Engineering and Knowledge Engineering (IJSEKE)*, 29(10):1403–1438.

[Marques et al. 2021] Marques, J., Yelisetty, S., and Barros, L. (2021). Um mapeamento sistemático da literatura no uso da iec 62304. *Journal of Health Informatics*.

[Mauer and Marin 2017] Mauer, T. and Marin, H. (2017). Instrumento de avaliação de implantação de sistemas de informação em saúde. *Journal of Health Informatics*, 9(4):111–118.

[Mchugh et al. 2012] Mchugh, M., Caffery, F. M., and Casey, V. (2012). Software process improvement to assist medical device software development organizations to comply with the amendments to the medical device directive. *IET Software*, 6(5):431–437.

[Munch et al. 2012] Munch, J., Armbrunt, O., Kowalczyk, M., and Soto, M. (2012). *Software Process Definition and Management*. Springer-Verlag, Berlim, Germany.

[Palmer and Felsing 2002] Palmer, S. R. and Felsing, J. M. (2002). *A Practical Guide to Feature Driven Development*. Pearson Educational.

[Peterson et al. 2015] Peterson, K., Vakkalanka, S., and Kuzniarz, L. (2015). Guidelines for conducting systematic mapping studies in software engineering: An update. *Information and Software Technology*, 64:1–18.

[Pressman and Maxim 2015] Pressman, R. and Maxim, B. (2015). *Software Engineering: A Practitioner's Approach*. McGraw-Hill Education.

[Regan et al. 2013] Regan, G., Caffery, F. M., Daid, K. M., and D. Flood, D. (2013). Medical device standards' requirements for traceability during the software development lifecycle and implementation of a traceability assessment model. *Computer Standards Interfaces*, 36(1):3–9.

[Rust et al. 2016] Rust, P., Flood, D., and McCaffery, F. (2016). Creation of an iec 62304 compliant software development plan. *Journal of Software Evolution and Process*, 28(11):1.1–1.10.

[Schwaber and Beedle 2001] Schwaber, K. and Beedle, M. (2001). *Agile Software Development with SCRUM*. Prentice-Hall.

[Sommerville 2015] Sommerville, I. (2015). *Software Engineering*. Pearson.

[Stapleton 1997] Stapleton, J. (1997). *DSDM - Dynamic Systems Development Method*. Addison-Wesley.

[Stober and Hansmann 2010] Stober, T. and Hansmann, U. (2010). *Agile Software Development - Best Practices for Large Software Projects*. Springer.

[Sutherland 2010] Sutherland, J. (2010). *SCRUM Handbook,*. Scrum Training Institute Press.

[Tsui et al. 2015] Tsui, F., Karam, O., and Bernal, B. (2015). *Essentials of Software Engineering*. Jones Bartlett Learning.

[Varri and de la Cruz 2019] Varri, A. and de la Cruz, P. K.-Z. R. (2019). Software life cycle standard for health software. *Stud Health Technol Inform*, 264:868–872.

[Vuori 2011] Vuori, M. (2011). Agile development of safety-critical software. Technical report, Tampere University of Technology.

[Wasson 2015] Wasson, C. (2015). *System Engineering Analysis, Design, and Development: Concepts, Principles, and Practices*. Wiley Series in Systems Engineering and Management.

[Wong and Callaghan 2012] Wong, K. and Callaghan, C. (2012). Managing requirements baselines for medical device software development. In *2012 IEEE International Systems Conference (SysCon)*.