4.2.2 Usability

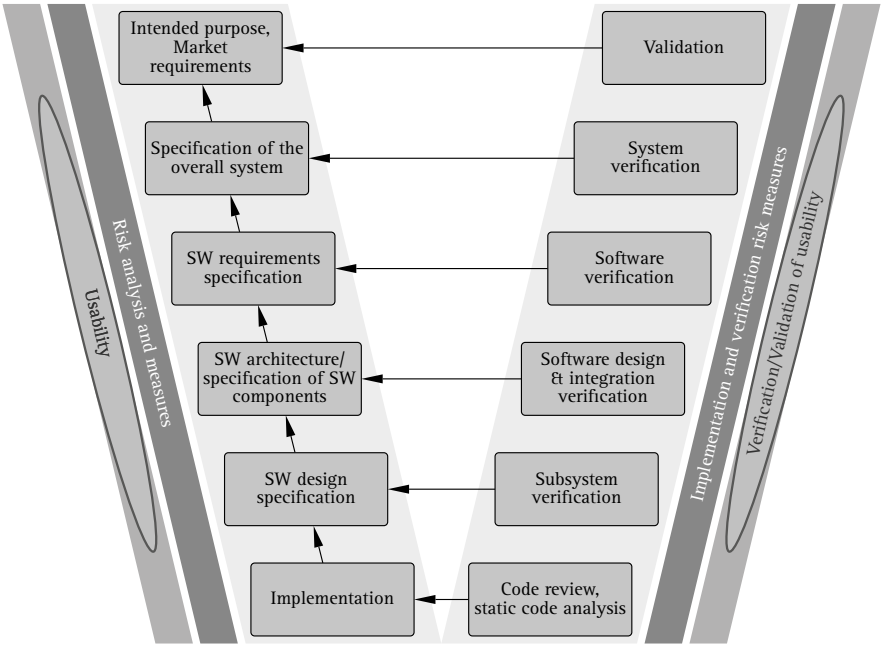

Figure 4.3 Verification/validation of the usability, SW = software

Applying usability includes not only specification and design activities (left side of the V-model) but also verification/validation activities (right side of the V-model).

Motivation

In general, it is essential to involve the users and their needs as early as possible to develop the right product. Therefore, good usability will directly affect user satisfaction and the economic success of the product in the market.

In medical practice, monitoring and treatment of patients increasingly relies on medical devices. User errors arising from the insufficient usability of electrical medical devices are a frequent source of risks for patients and users. Thus, usability is closely related to risk management and designed to minimize user errors and risks stemming from use. Some, but not all, types of misuse can be controlled this way.

Apart from the primary objective of implementing the safety-relevant requirements in the design, there is also the potential, based on usability testing, to come up with alternative solutions for training and process design.

Philosophy of User Interface Design

One suitable approach to the design of interactive products is the User Centered Design methodology according to DIN ISO 9241-210, which represents a structured iterative procedure for designing user interfaces. All activities are centered around the users with their goals, needs and characteristics.

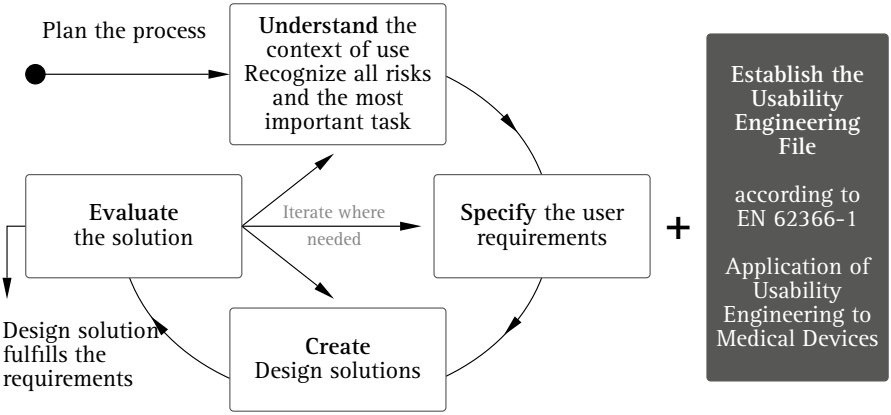

Figure 4.4 User Centered Design Process

After planning the complete process, the latter is implemented via several detailing loops. Which of the several techniques for determining the usability will be used depends on the need and detailing level.

Understanding the context of usage:

- Interviews, use of questionnaires, checklists
- Observation/support of the users in the future application environment
- Personas methodology
- Analysis of competitive and related products

Specification of the user requirements:

- Use Cases
- Scenarios
- UI specifications

Creating design solutions

- Paper drafts
- Wireframes
- Software click-prototypes

Evaluating the solution

- Formative user tests of prototypes at various levels of maturity through expert reviews, usability walk-throughs, user tests, field tests
- Validation of the medical device in user tests, field tests, clinical trials

From User Needs to Visual Design

James Jesse Garret defined the elements and levels of user experience. These are illustrated in the following figure.

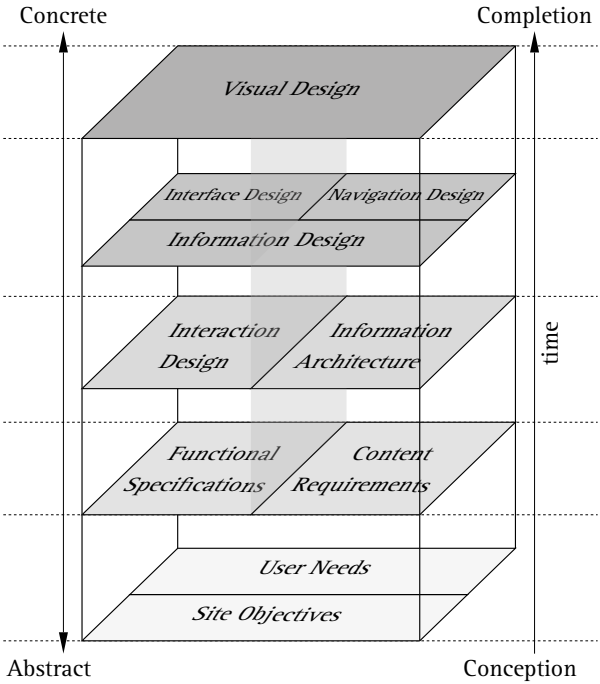

Figure 4.5 Elements of user experience according to *James Jesse Garret* (source: www.jjg.net/ia)

Unified operation of one or more interactive products is achieved primarily through consistency in the three upper elements of the user experience.

- **Design of the interaction and information architecture:** This level requires consistent application of the user interface patterns and rules of interaction. These are laid down in style guides to ensure a consistent application.
- **Information Design:** Consistent presentation and arrangement of uniform information, infographics, navigation elements and other recurrent elements is important. Uniform information should always be depicted in identical mode of presentation and interaction.
- **Visual design of the user interface:** A unified look and feel of all features belonging to the product and the user interfaces of a product family ensures a high degree of recognition with the user.

Usability file

The usability engineering results must be documented in the usability file. Type and scope of the usability engineering may vary, depending on the type of medical device, and should be adapted based on safety. The records and other documents, which together constitute the usability file, may become part of other documents and files, e.g., the product master file of the marketing authorization holder or the safety management file.

4.2.3 System architecture/specification of the software components

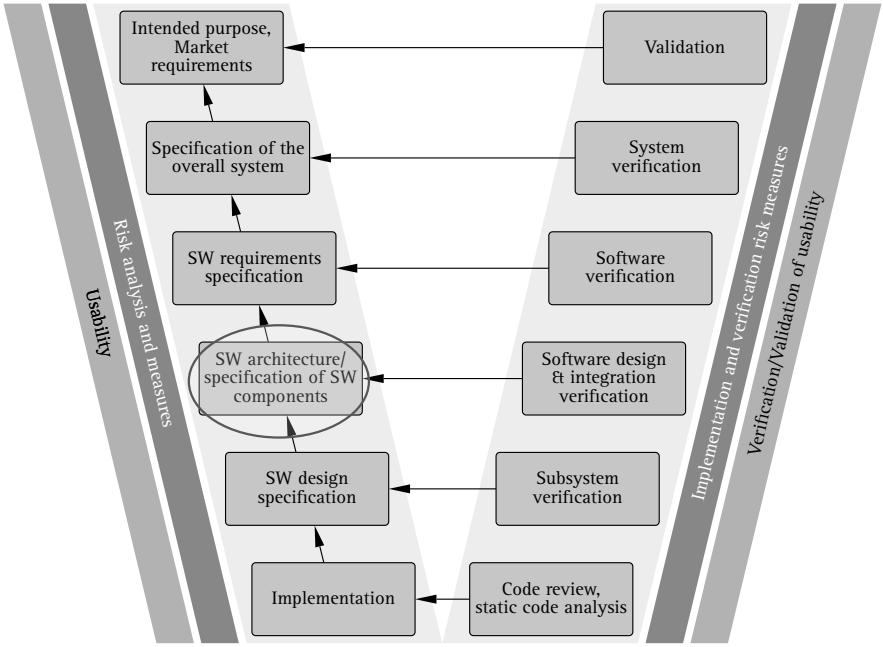

Figure 4.6 System architecture and specification of the components, SW = software

Methodology

Development of the system architecture relies on an iterative/incremental methodology. After having created an initial draft version of the system architecture, the former is updated and upgraded with continuous findings and activities in the areas of requirement management and software design. However, the goal is to create and release a stable first version of the system architecture early in the project.

When creating the software architecture (in this section used synonymously with system architecture) it is important to identify the drivers of the architecture. These are requirements from various areas with a particularly large impact on the software architecture. This does not just include the non-functional requirements (e.g., performance, testability, usability, and technical feasibility). Based on these

requirements, the software architecture is created, assessed, and adapted, as well as extended through further iteration.

One important aspect of software architecture is the identification of software of unknown provenance (SOUP)/commercial-off-the-shelf (COTS) software and their integration into the architecture (also see section 4.3.4.1).

Software architecture and risk management

The risk management findings must become part of the software architecture early on. To this end, initial considerations of possible threats and causes should be addressed and with regard to the software architecture, strategic, conceptual or technical solutions should be developed and described in order to develop a safe medical device. At the software architecture levels, considerations and solutions for a safe medical device might be conceptual descriptions for the following areas of example:

- Ensuring data integrity
- Secure communication channels/protocols
- Safeguarding of algorithms/calculations
- Process control/monitoring

Description of the software architecture

The software documentation should provide an understandable summary of the software system. The 4+1 architectural view model by *Philippe Kruchten* is well suited for this.

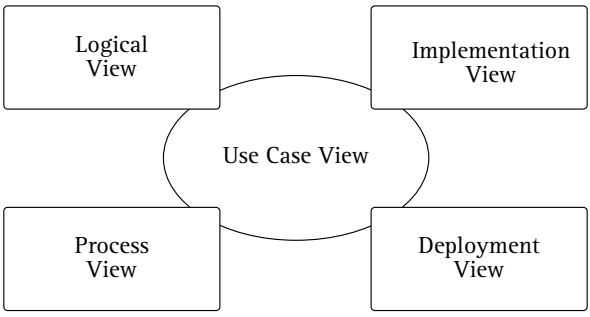

Figure 4.7 4+1 architectural view model according to *Kruchten*

The four views serve to look at the system from different perspectives and describe the viewpoints of different stakeholders.

- **Logical view***:* In the Logical View the object model of the software is described with the focus on system functionality. Usually, this relies on Unified Modeling Language (UML) diagrams, e.g., class diagrams and communication diagrams.
- **Development view:** In the Development View the static structure of the software is described from the viewpoint of the developer (e.g., layer, software management, packages ...) Usually, this relies on UML diagrams such as component diagrams and package diagrams.
- **Process view:** The Process View is concerned with the dynamic aspects of the system, such as states and transitions, synchronization, flow of message sequences, concurrency, scheduling or real-time constraints. Usually, this relies on UML diagrams, e.g., state chart diagrams, sequence diagrams and activity diagrams.
- **Physical view***:* The Physical View, also known as Deployment View, describes the deployment of the software on various hardware components and the communication between these components. Usually, this relies on the UML deployment diagram.
- **Use Case view:** The Use Case View describes the functional requirements of the system (here as use cases). In the present case, this view has already been described by the documentation of the software requirements specification.

Apart from these views it helps to expand the architecture documentation with a chapter on decisions. Especially in fundamental/critical decisions it is quite helpful to describe not only the chosen solution, but also the alternatives considered during the decision-making process, and to also document the reasons for the decision.

Regulatory aspects

From the regulatory point of view the software architecture must address the following issues:

- The architecture must meet the software requirements.
- The architecture must describe the software components and the interfaces between them.
- The SOUP/COTS components must be identified and specified, including the performance requirements as well as the requirements for hardware and software.
- The documentation must describe the impact of risk management measures on the software architecture.

4.2.4 Detailed software design

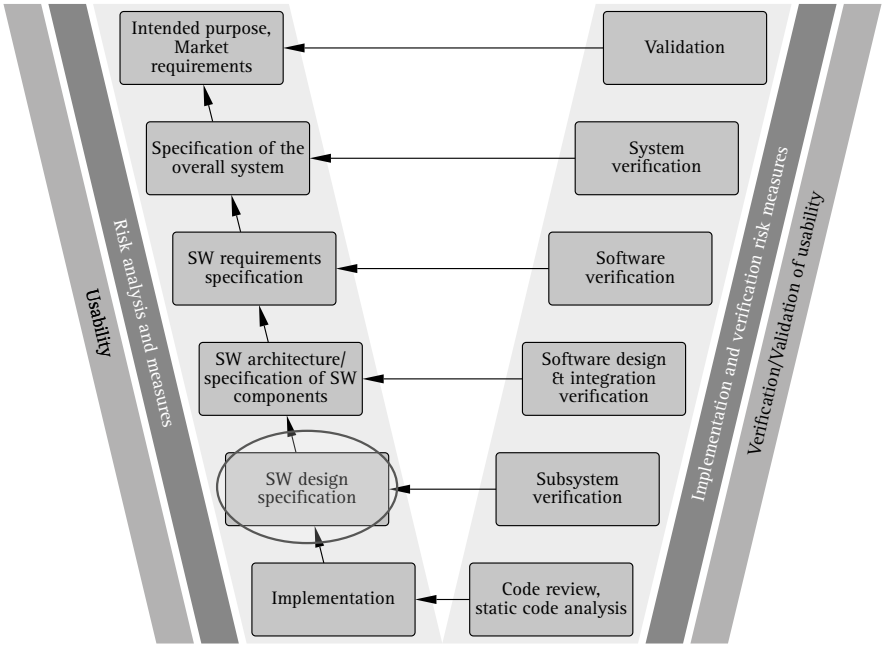

Figure 4.8 Detailed software design, SW = software

Depending on the safety classification of the software, a detailed design could/should subdivide the software further.

The IEC 62304-1 standard suggests the following differentiation for system steps from large to small "system → component → unit", with the proviso that the manufacturer decides on how the subdivision will be used.

It should be remembered that, due to certain reasons to be clarified by the manufacturer, the latter may decide not to further subdivide special software systems. However, in the high-security software class (C) forgoing a detailed design will be hard to argue because the detailed design clearly defines internal relationships and supports detailed testing.

Provided that the software system has been detailed down to the level of the software unit, a detailed design must be documented for the appropriate unit to support the implementation (and also the testing in terms of the validation).

Particular attention should be paid to the unit interfaces. These considerations should include interfaces to other units as well as hardware interfaces.

Verification of the detailed design

Like all specifications, the detailed design of the units must be subjected to verification (along the lines of a review).

As part of such a verification it should be ensured that

- the detailed design implements the software architecture (described above) and
- that there are no conflicts with the software architecture.

The execution of this verification and its results must be documented.

4.2.5 Implementation

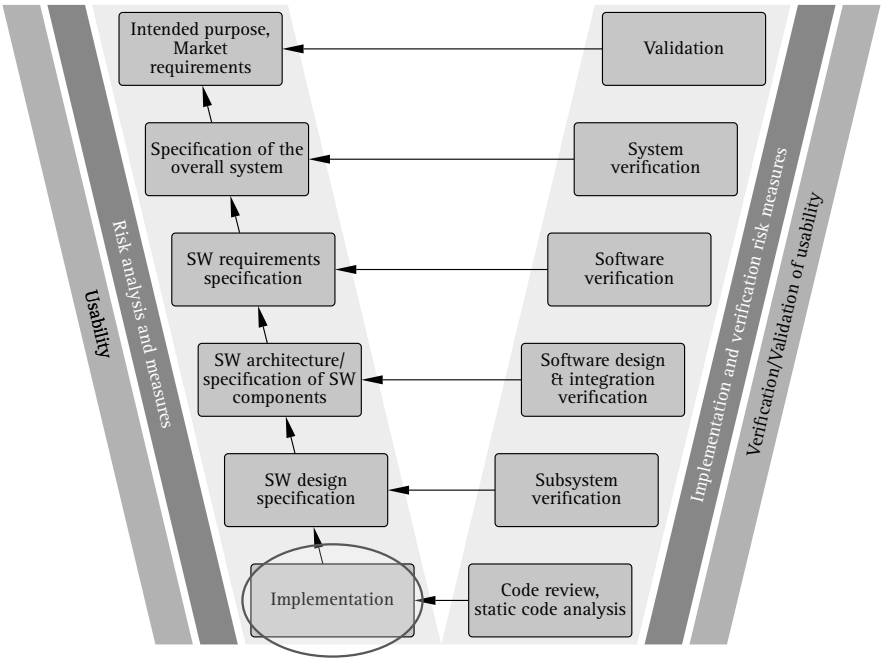

Figure 4.9 Software implementation, SW = software

The software must evidently be implemented irrespective of the safety classification selected for the software. The implementation step represents the transition from the "paper world" (corresponding with the left side of the V-model) to the real product (right side).

However, there are some provisions which should be noted.

If the detailed design has subdivided the software system into units, these will have to be verified (along the lines of testing) as part of the implementation.

Note: The verification workflow and the various approaches to testing are described in section 4.2.7.

At this point the different focus of each test approach should be emphasized:

- In unit testing the goal is to prove that the implemented code is working, e.g., that a described algorithm delivers the expected result.
- In system testing the goal is to prove that the software system fulfills the requirements. This goes beyond unit testing, which only deals with a small part of the functionalities but tests these rather intensely.

4.2.6 Risk management

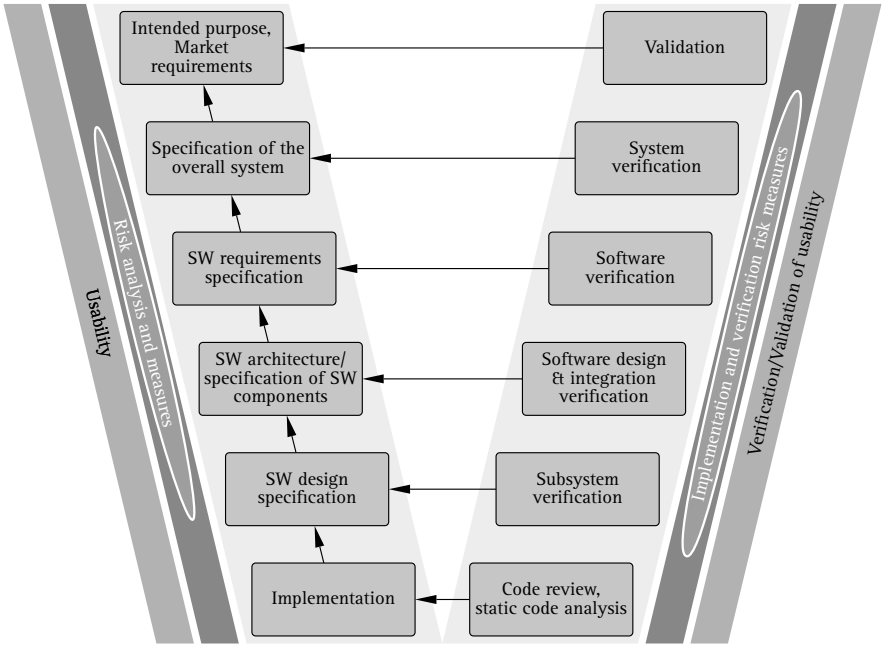

Figure 4.10 Risk management, SW = software

Risk management includes activities in the areas of specification and design (left side of the V-model), as well as in verification/validation (right side of the V-model).

4.2.6.1 General

The risk management system described here is based on the European standard EN ISO 14971. This standard EN ISO 14971 specifies the area of application as follows:

"This international standard specifies a process for a manufacturer to identify the hazards associated with medical devices, including *in vitro* diagnostic (IVD) medical devices, to estimate and evaluate the associated risks, to control these risks, and to monitor the effectiveness of the controls [...]."

It identifies several areas of application for risk management. Before activities are carried out, the focus to be looked at must be clearly defined.