4.2.2 Usability/Gebrauchstauglichkeit

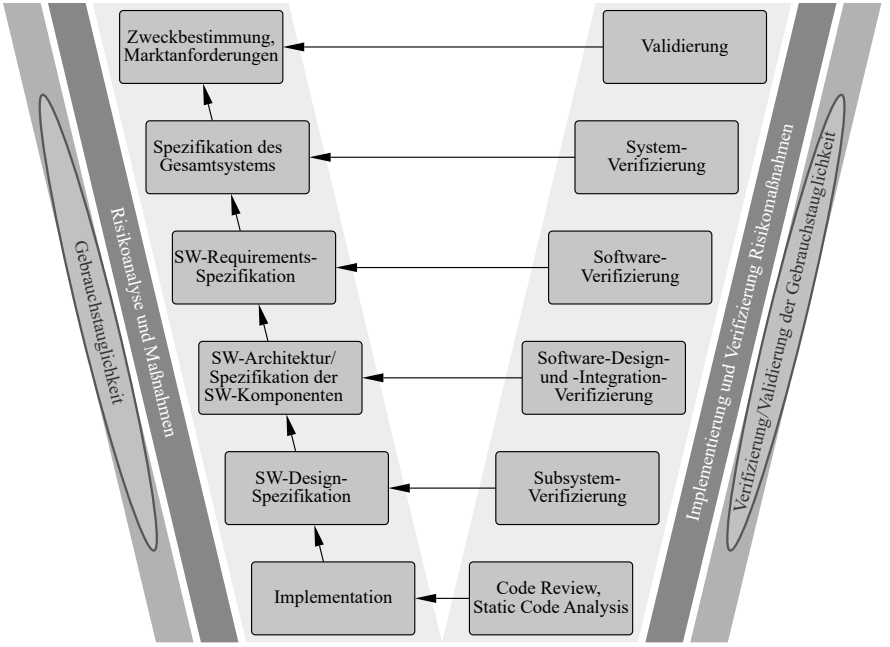

Bild 4.3 Verifizierung/Validierung der Gebrauchstauglichkeit – SW Software

Die Anwendung der Gebrauchstauglichkeit beinhaltet sowohl Spezifikations- und Design-Aktivitäten (linke Seite des V-Modells) als auch Verifizierungs-/Validierungsaktivitäten (rechte Seite des V-Modells).

Motivation

Generell sind eine frühzeitige Einbindung der Anwender und deren Bedürfnisse essenziell wichtig, um das richtige Produkt zu entwickeln. Eine gute Gebrauchstauglichkeit beeinflusst somit direkt die Zufriedenheit der Anwender und den wirtschaftlichen Erfolg des Produkts auf dem Markt.

In der medizinischen Praxis werden in zunehmendem Maße medizinische Geräte für die Überwachung und Behandlung von Patienten verwendet. Benutzungsfehler, verursacht durch unzureichende Gebrauchstauglichkeit von medizinischen elektrischen Geräten, sind eine häufige Quelle für Patienten- und Anwendergefährdungen. Gebrauchstauglichkeit ist also eng mit dem Risikomanagement verknüpft und soll

ihrerseits Benutzungsfehler und mit dem Gebrauch verbundene Risiken minimieren. Einige, aber nicht alle Arten falschen Gebrauchs können so beherrscht werden.

Neben dem vorrangigen Ziel, die Safety-relevanten Anforderungen im Design zu implementieren, bestehen gleichfalls Möglichkeiten, basierend auf Gebrauchstauglichkeitsuntersuchungen, alternative Lösungen im Bereich Schulung und Prozessgestaltung zu finden.

Philosophie bezüglich *User Interface Design*

Ein geeigneter Ansatz zur Gestaltung von interaktiven Produkten ist der *User Centered Design* Ansatz nach DIN ISO 9241-210, der ein strukturiertes und iteratives Vorgehen beim Entwurf von Benutzerschnittstellen darstellt. Im Zentrum aller Aktivitäten steht dabei der Anwender mit seinen Zielen, Bedürfnissen und Eigenschaften.

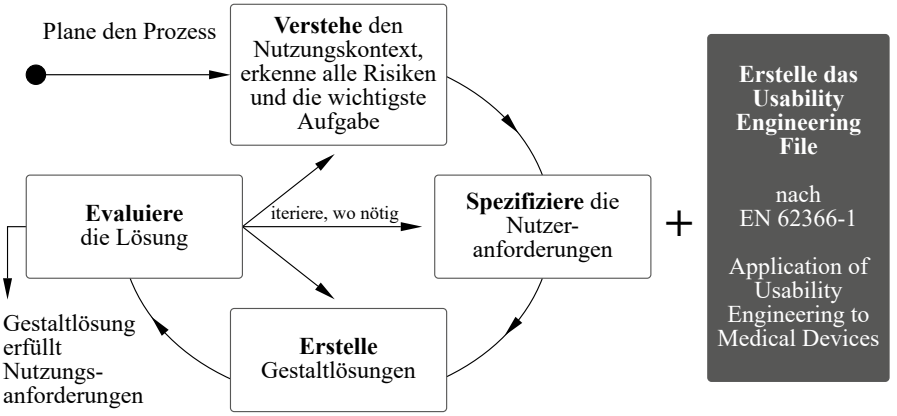

Bild 4.4 User Centered Design Process

Der gesamte Prozess muss geplant werden und wird dann in mehreren Detaillierungsschleifen durchgeführt. Dabei können je nach Bedarf und Detaillierungsebene verschiedene Methoden zur Erfassung der Gebrauchstauglichkeit zum Einsatz kommen.

Verstehen des Nutzungskontexts:

- Interviews, Einsatz von Fragebögen, Checklisten,
- Beobachtung/Begleitung der Anwender in der späteren Anwendungsumgebung,
- Personasmethodik,
- Analyse von Konkurrenzprodukten oder verwandten Produkten.

Spezifizieren der Anforderungen der Anwender:

- *Use Cases*,
- Szenarien,
- UI-Spezifikationen.

Erstellen von Gestaltungslösungen:

- Papierskizzen,
- *Wireframes*,
- Software-Klick-Prototypen.

Evaluieren der Lösung:

- Formative Anwendertests von Prototypen verschiedener Reifegrade durch *Expert Reviews, Usability Walkthroughs,* Anwendertests, Feldtests,
- Validierung des Medizinprodukts in Anwendertests, Feldtests, klinischen Studien.

Von *User Needs* zum *Visual Design*

James Jesse Garret definiert die Elemente und Ebenen der *User Experience.* Diese sind in **Bild 4.5** dargestellt.

Eine einheitliche Bedienung eines oder mehrerer interaktiver Produkte wird vor allem über eine Konsistenz in den drei oberen Elementen der *User Experience* erreicht.

- Gestaltung der Interaktion und Informationsarchitektur: Auf dieser Ebene muss die konsequente Anwendung von *User Interface Patterns* und Interaktionsregeln beachtet werden. Diese werden in *Styleguides* festgehalten, um eine durchgehende Anwendung zu gewährleisten.
- *Information Design:* Wichtig ist dabei eine konsistente Darstellung und Anordnung von gleichartigen Informationen, Infografiken, Navigationselementen und anderen wiederkehrenden Elementen. Gleichartige Informationen sollten immer mit der gleichen Darstellungs- und Interaktionsform präsentiert werden.
- Visuelle Gestaltung der Oberfläche: Ein einheitliches *Look-and-Feel* über alle zum Produkt zugehörigen Features oder zu einer Produktfamilie gehörenden *User Interfaces* gewährleisten beim Anwender einen hohen Wiedererkennungswert.

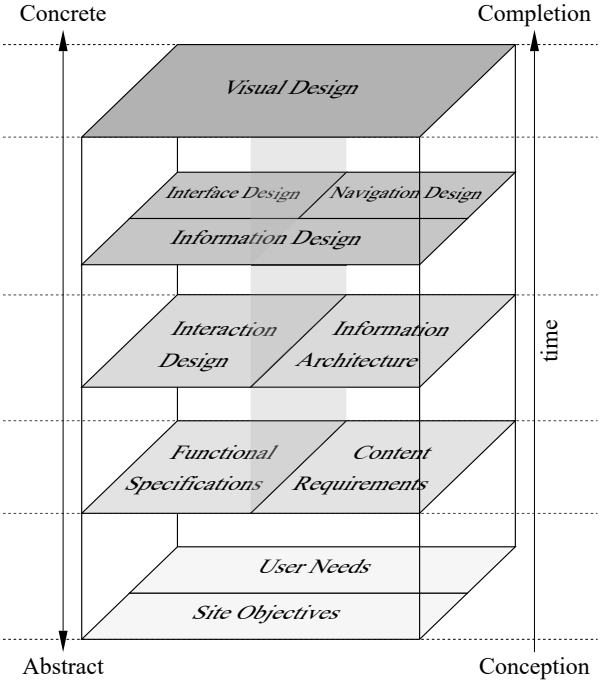

Bild 4.5 Elemente der User Experience von *James Jesse Garret* (Quelle: www.jjg.net/ia)

Gebrauchstauglichkeitsakte

Die Ergebnisse des *Usability Engineering* müssen in der Gebrauchstauglichkeitsakte aufgezeichnet werden. Das *Usability Engineering* kann in Form und Umfang variieren, abhängig von der Art des Medizinprodukts und sollte Safety-getrieben angepasst werden. Die Aufzeichnungen und andere Dokumente, welche die Gebrauchstauglichkeitsakte bilden, können Teil anderer Dokumente und Akten sein, z. B. der Produktakte des Inverkehrbringers oder des Risk-Management-File (Risikomanagement-Akte).

4.2.3 Systemarchitektur/Spezifikation der Software-Komponenten

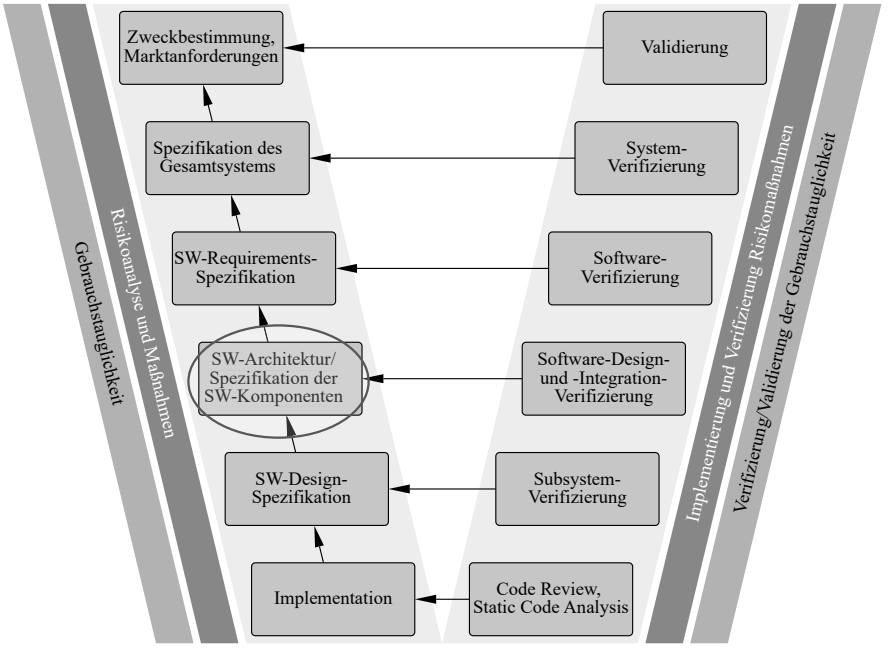

Bild 4.6 Software-Architektur und Spezifikation der Komponenten – SW Software

Methodik

Die Methodik bei der Erstellung der Systemarchitektur ist eine iterative/inkrementelle. Die Systemarchitektur wird in einer ersten *Draft Version* erstellt und mit fortlaufenden Erkenntnissen und Aktivitäten im Bereich Anforderungsmanagement und Software Design aktualisiert und erweitert. Ziel ist es jedoch, in einer frühen Projektphase eine stabile erste Version der Software-Architektur zu erstellen und freizugeben.

Wichtig bei der Erstellung der Software-Architektur (in diesem Abschnitt synonym zur Systemarchitektur verwendet) ist die Identifikation von Architekturtreibern. Dies sind Anforderungen aus verschiedenen Bereichen, die einen besonders großen Einfluss auf die Software-Architektur haben. Dazu gehören insbesondere die nicht funktionalen Anforderungen (z. B. *Performance,* Testbarkeit, Gebrauchstauglichkeit, technische Machbarkeit). Auf Basis dieser Anforderungen wird eine Software-Architektur erstellt, bewertet und in den weiteren Iterationen angepasst und erweitert.

Ein wichtiger Aspekt der Software-Architektur ist die Identifikation von *Software of unknown provenance* (SOUP)/*Commercial-off-the-shelf*-(COTS-)Software und die Einbindung dieser in die Architektur (siehe auch Abschnitt 4.2.3).

Software-Architektur und Risikomanagement

Die Erkenntnisse aus dem Risikomanagement müssen frühzeitig in die Software-Architektur einfließen. Dazu sollten erste Betrachtungen zu den möglichen Gefährdungen und Ursachen aufgegriffen werden und hinsichtlich der Software-Architektur strategische, konzeptionelle bzw. technische Lösungen erarbeitet und beschrieben werden, welche darauf abzielen, ein sicheres Medizinprodukt zu entwickeln. Überlegungen und Lösungen zu einem sicheren Medizinprodukt auf Ebenen der Software-Architektur können konzeptionelle Beschreibungen zu folgenden, beispielhaften Bereichen sein:

- Sicherstellung von Datenintegrität,
- Sichere Übertragungswege/-protokolle,
- Absicherung von Algorithmen/Berechnungen,
- Steuerung/Überwachung von Prozessen.

Beschreibung der Software-Architektur

Die Dokumentation der Software-Architektur soll einen verständlichen Überblick über das Software-System bieten. Dazu eignet sich das 4+1-Sichtenmodell von *Philippe Kruchten*.

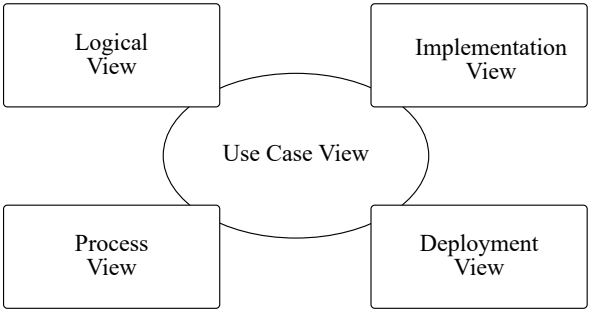

Bild 4.7 4+1-Sichtenmodell nach *Kruchten*

Die vier Sichten werden benutzt, um das System aus verschiedenen Aspekten zu betrachten und die Blickwinkel verschiedener Stakeholder zu beschreiben.

- *Logical View:* Im *Logical View* wird das Objektmodell der Software mit dem Fokus auf die Systemfunktionalität beschrieben. Dazu werden meistens *Unified Modeling Language* (UML) Diagramme wie z. B. Klassendiagramme oder Kommunikationsdiagramme verwendet.
- *Development View:* Im *Development View* wird die statische Struktur der Software aus Sicht des Entwicklers beschrieben (z. B. *Layer,* Software-Management, Pakete …). Dazu werden meist UML-Diagramme wie z. B. Komponentendiagramme oder Paketdiagramme verwendet.
- *Process View:* Der *Process View* beschäftigt sich mit den dynamischen Aspekten des Systems wie z. B. Zustände und Übergänge, Synchronisation, Ablauf von Nachrichtensequenzen, *Concurrency, Scheduling* oder *Real-time*-Randbedingungen. Dazu werden meist UML-Diagramme wie z. B. *Statecharts,* Sequenzdiagramme oder Aktivitätsdiagramme verwendet.
- *Physical View:* Der *Physical View* oder auch *Deployment View* beschreibt die Verteilung der Software auf verschiedene Hardware Komponenten und die Kommunikation zwischen den Komponenten. Dazu wird meist das UML-Verteilungsdiagramm verwendet.
- *Use Case View:* Der *Use Case View* beschreibt die funktionalen Anforderungen des Systems (hier in der Form von *Use Cases).* Im hier beschriebenen Fall ist dieser *View* bereits über die Dokumentation der *Software-Requirements*-Specification beschrieben.

Zusätzlich zu den genannten *Views* ist es sinnvoll, in der Architekturdokumentation ein Kapitel mit Entscheidungen einzufügen. Gerade bei grundlegenden/kritischen Entscheidungen ist es sehr nützlich, nicht nur die gewählte Lösung zu beschreiben, sondern auch die im Entscheidungsprozess betrachteten Alternativen und die Gründe für die Entscheidung mitzudokumentieren.

Regulatorische Aspekte

Aus regulatorischer Sicht muss die Software-Architektur folgende Punkte enthalten:

- Die Architektur muss die Anforderungen an die Software abbilden.
- Die Architektur muss die Software-Komponenten und die Schnittstellen zwischen diesen beschreiben.
- Die SOUP/COTS Komponenten müssen inkl. Leistungsanforderungen und Anforderungen an die Hard- und Software identifiziert und spezifiziert werden.
- Der Einfluss von Risikobeherrschungsmaßnahmen auf die Software-Architektur muss beschrieben werden.

4.2.4 Detailliertes Software-Design

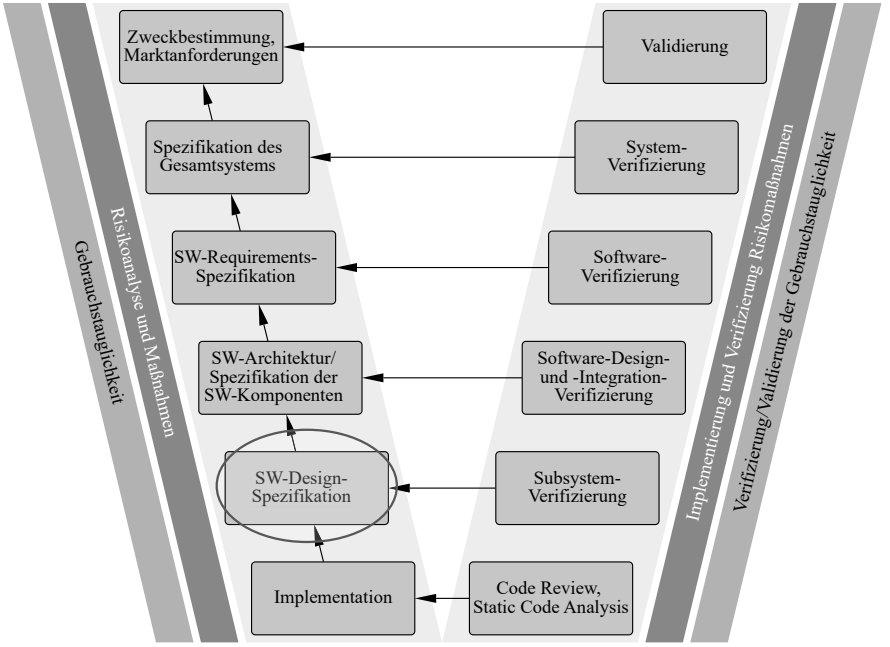

Bild 4.8 Detailliertes Software-Design – SW Software

In Abhängigkeit von der Software-Sicherheitsklassifizierung kann/sollte das Software-System in einem detaillierten Design weiter unterteilt werden.

Die Norm IEC 62304-1 schlägt zur Unterteilung vom Großen zum Kleinen die Stufen System → Komponente → Einheit (englisch: *system → component → unit)* vor, unter dem Vorbehalt, dass der Hersteller entscheidet, wie diese Unterteilung angewendet wird.

Dabei muss berücksichtigt werden, dass der Hersteller aus bestimmten und von ihm dann darzulegenden Gründen entscheiden kann, spezielle Software-Systeme nicht weiter zu unterteilen. Die Argumentation, auf ein detailliertes Design zu verzichten, wird allerdings in der hohen Software-Sicherheitsklasse (C) schwierig zu führen sein, denn durch das detaillierte Design werden sowohl die internen Zusammenhänge klar definiert als auch das detaillierte Testen unterstützt.

Sofern das Software-System auf die Ebene „Software-Einheit" („*Software-Unit*") herunter gebrochen ist, so muss für die jeweilige Unit ein detailliertes Design do-

kumentiert werden, um die Implementierung (und auch das Testen im Sinne der Verifizierung) zu unterstützen.

Besonderes Augenmerk ist dabei auf die Schnittstellen der Units zu legen. Diese Betrachtungen sollen sich sowohl auf Schnittstellen zu weiteren Units beziehen als auch Hardware-Schnittstellen berücksichtigen.

Verifizierung des detaillierten Designs

Wie alle Vorgaben ist auch das detaillierte Design der Units einer Verifizierung (im Sinne eines Reviews) zu unterziehen.

Im Rahmen einer solchen Überprüfung soll sichergestellt werden, dass

- das detaillierte Design die (weiter oben beschriebene) Software-Architektur implementiert und
- keine Widersprüche zur Software-Architektur bestehen.

Die Durchführung dieser Verifizierung sowie die Ergebnisse sind zu dokumentieren.

4.2.5 Implementierung

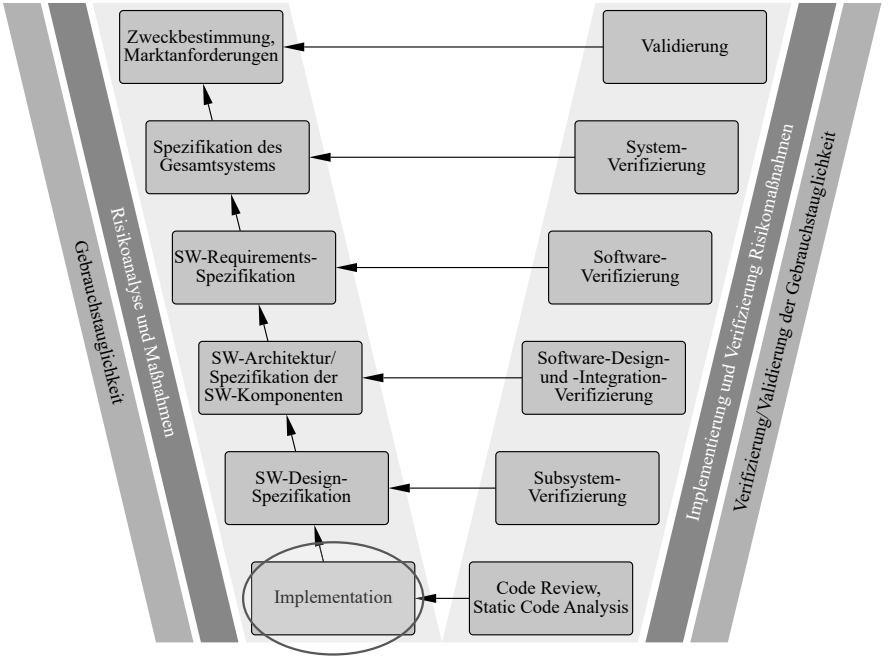

Bild 4.9 Software-Implementierung – SW Software

Augenscheinlich ist die Software zu implementieren, unabhängig von der gewählten Software-Sicherheitsklassifizierung. Der Schritt der Implementierung stellt doch den Übergang zwischen der „Papierwelt“ (entsprechend der linken Seite des V-Modells) und dem realen Produkt (rechte Seite) dar.

Dennoch sind einige Vorgaben zu beachten.

Sofern das Software-System im detaillierten Design in Einheiten (Units) zerlegt wurde, ist im Rahmen der Implementierung die Verifizierung (im Sinne von „Testen“) dieser Units vorzusehen.

Anmerkung: Die Vorgehensweise der Verifizierung sowie die unterschiedlichen Testansätze sind in Abschnitt 4.2.7 beschrieben.

Wichtig an dieser Stelle ist die Unterscheidung des Fokus des jeweiligen Testansatzes:

- Gegenstand der Unit-Tests ist der Nachweis, dass der implementierte Code funktioniert, also beispielsweise ein beschriebener Algorithmus das erwartete Ergebnis liefert.
- Gegenstand der System-Tests ist, dass die Anforderungen an das Software-System erfüllt werden. Dies geht über das Maß des Unit-Tests hinaus, die sich nur mit einem kleinen Umfang an Funktionalität befassen, aber mit diesem sehr intensiv.

4.2.6 Risikomanagement

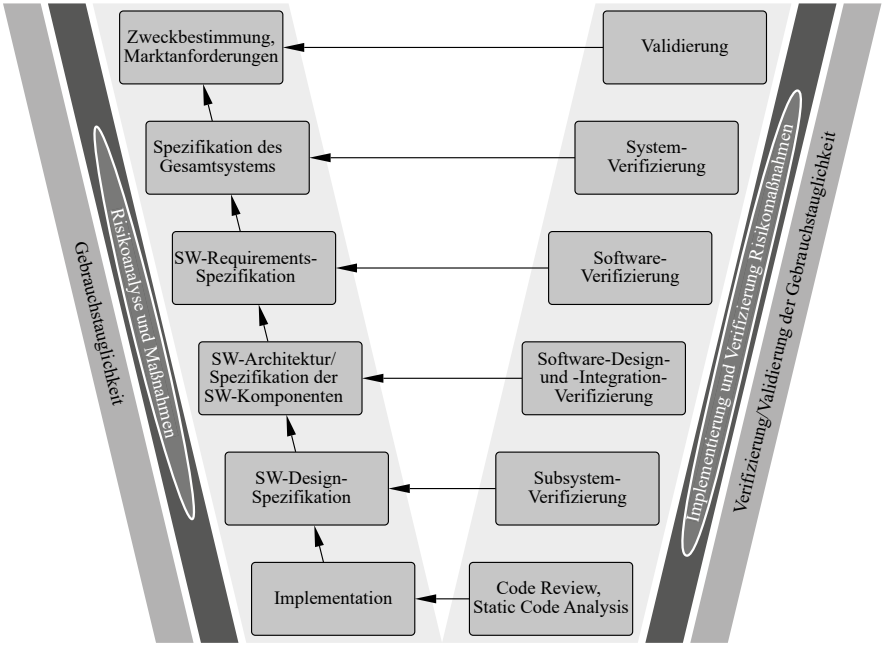

Bild 4.10 Risikomanagement – SW Software

Das Risikomanagement beinhaltet Aktivitäten sowohl im Bereich Spezifikationen und Design (linke Seite des V-Modells) als auch im Bereich Verifizierung/Validierung (rechte Seite des V-Modells).