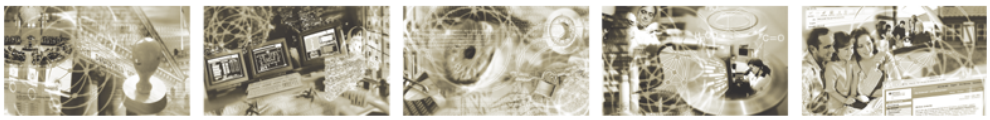
Bundesamt
für Sicherheit in der
Informationstechnik



Band M, Kapitel 9: Software

Bundesamt für Sicherheit in der Informationstechnik
Postfach 20 03 63
53133 Bonn
Tel.: +49 22899 9582-0
E-Mail: Hochverfuegbarkeit@bsi.bund.de
Internet: https://www.bsi.bund.de

Inhaltsverzeichnis

Tabellenverzeichnis

1 Software

Die nachfolgenden Maßnahmenkataloge beschreiben Verfahren und Lösungen für die Entwicklung und den Einsatz von hoch verfügbaren Softwarekomponenten. Die Strukturierung der Maßnahmen erfolgt in Form von Maßnahmenclustern für die am Lebenszyklus der Software ausgerichteten Subdomänen der HV-Domäne „Software“:

- Anforderungsdefinition und Design
- Entwicklung und Test
- Implementierung und Betrieb
- Notfall-Vorsorge und -Behandlung

1.1 Anforderungsdefinition & Design

<table>
<tr><th>Nr.</th><th>Maßnahmen</th></tr>
<tr><td>VM.9.1</td><td>
Bewertung des Reifegrades der Entwicklungsprozesse
Die Orientierung der Softwareentwicklung an einem prozessorientierten Referenzmodell ist Garant für die Ableitung der geforderten funktionalen und nicht funktionalen Eigenschaften aus den zuvor erhobenen Anforderungen. Für die Ermittlung und Bewertung der Qualität der Entwicklungsprozesse liegen etablierte Referenzmodelle (CMMI/SPICE) vor, die über Fähigkeits-/Reifegrade die Prozessqualität bewerten und einer Optimierung zugänglich machen. So können Fehler im Wirkbetrieb der Software vermieden und eine Verbesserung der Verfügbarkeit erreicht werden. Eine geeignete Möglichkeit um Transparenz hinsichtlich der Sicherheitseigenschaften von IT-Produkten zu schaffen, ist deren Prüfung und Bewertung nach einheitlichen Kriterien durch unabhängige Prüfstellen. Die geprüfte Sicherheitsleistung wird durch ein Zertifikat bestätigt. Als Maßstab zur Beurteilung der Sicherheit informationstechnischer Systeme dienen die Kriterien der ITSEC oder Common Criteria für die Prüfung und Bewertung der IT-Sicherheit.
<table>
<tr><th>Umsetzungsphase</th><th>Prinzip</th><th>Kriterien</th><th>wirkt gegen</th><th>Querverweis</th></tr>
<tr><td>Planung und Konzeption
Beschaffung</td><td>Zuverlässigkeit
Fehlertoleranz
Robustheit</td><td>Reifegrad
Fähigkeitsgrad
Funktionalität
Zuverlässigkeit</td><td>Menschliches Versagen
Fehler in Software</td><td>HVK (Software)
CobiT, PO8
CMMI
SPICE
http://www.bsi.bund.de
/zertifiz/zert/report.htm</td></tr>
</table>
</td></tr>
</table>

<table>
<tr><th>Nr.</th><th>Maßnahmen</th></tr>
<tr><td>VM.9.2</td><td>
Anforderungsdefinition
Die Anforderungsdefinition legt den Grundstein für die spätere Systemimplementierung und dient der Ermittlung der Systemanforderungen und der Umgebungen des zu entwickelnden Systems. In dieser Phase der Entwicklung ist es wichtig, die systemrelevanten Unternehmensprozesse zu verifizieren und zu analysieren, um spätere Fehler aufgrund falscher Annahmen und Ausgangsbedingungen zu verhindern.
<table>
<tr><th>Umsetzungsphase</th><th>Prinzip</th><th>Kriterien</th><th>wirkt gegen</th><th>Querverweis</th></tr>
<tr><td>Planung und Konzeption
Beschaffung</td><td>Priorisierung
Separation
Skalierbarkeit</td><td>Schnittstellen
Systemgrenzen</td><td>Menschliches Versagen
Sabotage, Manipulation</td><td>HVK (Software)</td></tr>
</table>
</td></tr>
<tr><td>VM.9.3</td><td>
Durchgängige Daten- und Parameterprüfung
Jedes Modul sowie jede Funktion innerhalb der Software ist für die Prüfung von eingehenden und ausgehenden Daten und Parameter selber verantwortlich. Es darf grundsätzlich nicht davon ausgegangen werden, dass übergeordnete Methoden oder Prozesse eine Fehlerfreiheit der Daten garantieren. Zur Gewährleistung der Korrektheit der Prüfungen muss für jedes Modul genau spezifiziert werden, welche Daten bzw. Parameter hinsichtlich ihres Formats und Inhalts als gültig zu gelten haben.
<table>
<tr><th>Umsetzungsphase</th><th>Prinzip</th><th>Kriterien</th><th>wirkt gegen</th><th>Querverweis</th></tr>
<tr><td>Planung und Konzeption
Beschaffung
Implementierung</td><td>Fehlertoleranz
Automatismen
Autonomie</td><td>Überwachung
Systemautonomie
Fehlertolerant
Fehlbedienungstolerant
Daten-/Parameterprüfung</td><td>Menschliches Versagen
Fehler in Hard- oder Software
Sabotage, Manipulation</td><td>HVK (Software)</td></tr>
</table>
</td></tr>
</table>

<table>
<tr><th>Nr.</th><th>Maßnahmen</th></tr>
<tr><td>VM.9.4</td><td>
Qualitätsbewertung nach DIN ISO 9126
Durch die Ermittlung und Bewertung der Softwarequalität anhand der in DIN ISO 9126 definierten Qualitätsmerkmale können Fehler im Wirkbetrieb der Software vermieden und so eine Verbesserung der Verfügbarkeit erreicht werden.
<table>
<tr><th>Umsetzungsphase</th><th>Prinzip</th><th>Kriterien</th><th>wirkt gegen</th><th>Querverweis</th></tr>
<tr><td>Planung und Konzeption
Beschaffung</td><td>Zuverlässigkeit
Fehlertoleranz
Separation
Priorisierung
Robustheit</td><td>Funktionalität
Zuverlässigkeit
(Reife, Fehlertoleranz, Robustheit, Konformität)
Benutzbarkeit
Effizienz
Änderbarkeit
Übertragbarkeit
Reifegrad</td><td>Menschliches Versagen
Fehler in Hard- oder Software</td><td>HVK (Software)
CobiT, PO8
DIN ISO 9126</td></tr>
</table>
</td></tr>
<tr><td>VM.9.5</td><td>
Bestimmung des Code-Quality-Index
Der Code-Quality-Index stellt ein definiertes Qualitätsmodell zur Verfügung, welches auf Basis des Forschungsprojektes „Qbench“ entwickelt wurde. Es beruht auf empirischem Material und leitet aus ausführlichen Untersuchungen bestehender Projekte Qualitätsindikatoren für konkrete Implementierungen ab. Durch die Bestimmung und Analyse des Code-Quality-Index können fehlerhafte Implementierungen erkannt und Verfügbarkeitsverluste vermieden werden.
<table>
<tr><th>Umsetzungsphase</th><th>Prinzip</th><th>Kriterien</th><th>wirkt gegen</th><th>Querverweis</th></tr>
<tr><td>Beschaffung
Implementierung</td><td>Transparenz
Fehlertoleranz</td><td>Code-Quality-Index</td><td>Menschliches Versagen
Fehler in Hard- oder Software</td><td>HVK (Software)</td></tr>
</table>
</td></tr>
</table>

<table>
<tr><th>Nr.</th><th colspan="5">Maßnahmen</th></tr>
<tr><td rowspan="3">VM.9.6</td><td colspan="5">Redundanz von Softwarekomponenten
Der Ausfall einer Softwarekomponente kann durch verschiedene Faktoren hervorgerufen werden. Durch die Verwendung redundanter Softwarekomponenten kann einem Ausfall entgegengewirkt werden, sofern es sich nicht um einen systematischen Fehler der Implementierung handelt.</td></tr>
<tr><td>Umsetzungsphase</td><td>Prinzip</td><td>Kriterien</td><td>wirkt gegen</td><td>Querverweis</td></tr>
<tr><td>Beschaffung
Implementierung
Betrieb</td><td>Redundanz</td><td>Aktivierungszeit
Grad
Aktivierung
Funktionell
Diversität
Hierarchie
Synchronität</td><td>Menschliches Versagen
Fehler in Hard- oder Software
Sabotage, Manipulation</td><td>HVK (Software)</td></tr>
</table>

Tabelle 1-1: Maßnahmenkatalog Software:Anforderungsdefinition & Design

1.2 Entwicklung und Test

<table>
<tr><th>Nr.</th><th>Maßnahmen</th></tr>
<tr><td>VM.9.7</td><td>
Entwicklungs- und Produktionsprozess
Der Einsatz eines geeigneten Modells zur Analyse und Bewertung des Entwicklungs- und Produktionsprozesses von Software wie z. B. CMMI erhöht die Zuverlässigkeit der Software und reduziert Anfälligkeiten der Implementierung, die auf Entwicklungsschwächen beruhen.
<table>
<tr><th>Umsetzungsphase</th><th>Prinzip</th><th>Kriterien</th><th>wirkt gegen</th><th>Querverweis</th></tr>
<tr><td>Planung und Konzeption
Beschaffung
Implementierung
Beschaffung</td><td>Fehlertoleranz
Skalierbarkeit
Separation
Robustheit</td><td>Komplexität
Designqualität
Wartbarkeit</td><td>Menschliches Versagen
Sabotage, Manipulation</td><td>HVK (Software)</td></tr>
</table>
</td></tr>
<tr><td>VM.9.8</td><td>
Verwendung von Standard-Architekturmustern
Architekturmuster definieren die fundamentalen Strukturen eines Softwaresystems. Zur Reduzierung der Komplexität der Systemarchitektur sowie zur Verbesserung der Wartbarkeit und Erweiterbarkeit ist die Verwendung von Architekturmustern im Softwaredesign empfehlenswert. Bei der Wahl des Architekturmodells sind die zu entwickelnden Softwaresysteme und deren Anwendungsgebiet zu berücksichtigen.
<table>
<tr><th>Umsetzungsphase</th><th>Prinzip</th><th>Kriterien</th><th>wirkt gegen</th><th>Querverweis</th></tr>
<tr><td>Planung und Konzeption
Beschaffung
Implementierung
Beschaffung</td><td>Fehlertoleranz
Skalierbarkeit
Separation
Robustheit</td><td>Komplexität
Designqualität
Wartbarkeit</td><td>Menschliches Versagen
Sabotage, Manipulation</td><td>HVK (Software)</td></tr>
</table>
</td></tr>
</table>

<table>
<tr><th>Nr.</th><th>Maßnahmen</th></tr>
<tr><td>VM.9.9</td><td>
Verwendung von Standard-Entwurfsmustern
Zur Reduzierung der Komplexität sowie zur Stabilitätsverbesserung der Systemarchitektur sollten beim Systemdesign nur Standard-Entwurfsmuster angewendet werden. Design-Patterns stellen Lösungsbeschreibungen für ein spezifisches Entwurfsproblem in Form von implementierungsuunabhängige und wiederverwendbaren Vorlagen dar. Bei der Verwendung von Design-Patterns sind der Anwendungsbezug sowie die Granularität der Muster zu berücksichtigen.
<table>
<tr><th>Umsetzungsphase</th><th>Prinzip</th><th>Kriterien</th><th>wirkt gegen</th><th>Querverweis</th></tr>
<tr><td>Planung und Konzeption
Beschaffung
Implementierung</td><td>Fehlertoleranz
Robustheit
Separation
Skalierbarkeit</td><td>Komplexität
Designqualität</td><td>Menschliches Versagen
Sabotage, Manipulation</td><td>HVK (Software)</td></tr>
</table>
</td></tr>
<tr><td>VM.9.10</td><td>
Design-Analyse mit Anti-Pattern
Als Anti-Pattern werden Entwurfsmuster bezeichnet, die auf ein schlechtes Softwaredesign hindeuten. Mit Hilfe von Anti-Pattern kann die Qualität des Software-Designs beurteilt und es können Implementierungsschwächen erkannt werden. Die Bewertung der Designqualität erlaubt die Auswahl und den Einsatz fehlertoleranter, robuster und zuverlässiger Software innerhalb von HV-Architekturen.
<table>
<tr><th>Umsetzungsphase</th><th>Prinzip</th><th>Kriterien</th><th>wirkt gegen</th><th>Querverweis</th></tr>
<tr><td>Beschaffung
Implementierung
Betrieb</td><td>Fehlertoleranz
Robustheit</td><td>Designqualität</td><td>Menschliches Versagen
Sabotage, Manipulation</td><td>HVK (Software)</td></tr>
</table>
</td></tr>
</table>

Nr.	*Maßnahmen*				
VM.9.11	**Fehlermaskierung** Als Fehlermaskierung wird ein Verfahren im Software-Design bezeichnet, bei dem ein Modul auf unterschiedliche Art und Weise von unabhängigen Teams realisiert wird. Diesen Modulen wird dann der sogenannte Maskierer vorgeschaltet, der mittels eines Ergebnisvergleichs fehlerhafte Resultate ausschließt, solange diese nicht in der Mehrheit der Module auftreten. Die bei diesem Verfahren realisierte Redundanz trägt zu Steigerung der Stabilität der Implementierung bei, da eine höhere Fehlertoleranz und Robustheit erreicht wird.				
	Umsetzungsphase	**Prinzip**	**Kriterien**	**wirkt gegen**	**Querverweis**
	Beschaffung Implementierung Betrieb	Fehlertoleranz Redundanz Robustheit	Fehlertolerant Automatisierungsgrad Systemautonomie Funktionell Information Diversität Fehlermaskierung	Menschliches Versagen Fehler in Hard- oder Software	HVK (Software)

Nr.	*Maßnahmen*
VM.9.12	**Fehlerkompensierung** Beim Verfahren der Fehlerkompensierung werden die Ergebnisse auf Fehler untersucht und durch ein nach gelagertes Modul wird das korrekte Ergebnis berechnet. Dieses erfordert eine entsprechende Redundanz der Ergebnisdaten. Beispiele hierfür sind Kommunikationsprotokolle bei denen Übertragungsfehler erkannt und korrigiert werden können. Dies führt zu einer deutlichen Verbesserung der Fehlertoleranz und Zuverlässigkeit von Softwaremodulen.

Umsetzungsphase	**Prinzip**	**Kriterien**	**wirkt gegen**	**Querverweis**
Beschaffung Implementierung Betrieb	Redundanz Fehlertoleranz Automatismen Robustheit	Fehlertolerant Automatisierungsgrad Reaktionszeit Systemautonomie Aktivierungszeit Zeit Funktionell	Menschliches Versagen Sabotage, Manipulation	HVK (Software)

<table>
<tr><th>Nr.</th><th>Maßnahmen</th></tr>
<tr><td>VM.9.13</td><td>
Fehlerbehebung
Beim Verfahren der Fehlerbehebung wird das Modul bei Erkennung eines fehlerhaften Ergebnisses in einen definierten Zustand zurückgesetzt und die Berechnung des Ergebnisses wird wiederholt. Dies führt im Falle von temporären Einflüssen zu einer Verbesserung der Robustheit.
<table>
<tr><th>Umsetzungsphase</th><th>Prinzip</th><th>Kriterien</th><th>wirkt gegen</th><th>Querverweis</th></tr>
<tr><td>Beschaffung
Implementierung
Betrieb</td><td>Automatismen
Fehlertoleranz
Robustheit</td><td>Fehlertolerant
Neustarts
Überwachung
Automatisierungsgrad
Reaktionszeit
Systemautonomie</td><td>Menschliches Versagen
Sabotage, Manipulation
Technische Ermüdung</td><td>HVK (Software)</td></tr>
</table>
</td></tr>
<tr><td>VM.9.14</td><td>
Pair-Programming
Beim Verfahren der Paarprogrammierung wird die Implementierung von zwei Entwicklern parallel an einem Arbeitsplatz vorgenommen. Hierbei ist jeweils ein Programmierer für das Schreiben des Codes verantwortlich, während der andere die Umsetzung prüft und mögliche Zusammenhänge analysiert. Diese Rollenverteilung wird regelmäßig getauscht. Dieses Verfahren führt zu einer höheren Codequalität und somit zu einer insgesamt besseren Verfügbarkeit der Software, da ein Programmierer in der Rolle des Beobachters Fehler und Risiken schneller erkennen kann.
<table>
<tr><th>Umsetzungsphase</th><th>Prinzip</th><th>Kriterien</th><th>wirkt gegen</th><th>Querverweis</th></tr>
<tr><td>Implementierung
Beschaffung</td><td>Fehlertoleranz
Redundanz
Robustheit
Separation</td><td>Überwachung
Organisatorisch
Personell
Funktionell</td><td>Menschliches Versagen
Fehler in Hard- oder Software</td><td>HVK (Software)</td></tr>
</table>
</td></tr>
</table>

<table>
<tr><th>Nr.</th><th>Maßnahmen</th></tr>
<tr><td>VM.9.15</td><td>Test-First-Programming
Ebenso wie das Pair-Programming entstammt das Verfahren des Test-First-Programming der Extreme Programming-Methode. Hier werden für neue Module zunächst geeignete Testmethoden entwickelt, die ein automatisiertes und wiederholbares Testen des Moduls ermöglichen. So kann vermieden werden, dass aufgrund menschlichen Versagens um Fehler „herum“ getestet wird. Die Robustheit und Zuverlässigkeit der Software kann so verbessert werden.
<table>
<tr><th>Umsetzungsphase</th><th>Prinzip</th><th>Kriterien</th><th>wirkt gegen</th><th>Querverweis</th></tr>
<tr><td>Beschaffung
Implementierung</td><td>Robustheit</td><td>Automatisierungsgrad
Überwachung
Fehlertolerant</td><td>Menschliches Versagen
Fehler in Hard- oder Software</td><td>HVK (Software)</td></tr>
</table></td></tr>
<tr><td>VM.9.16</td><td>Continous-Integration
Continuous Integration beschreibt ein Verfahren, bei der jede Änderung, die ein Entwickler an die zentrale Quellcodeverwaltung übergibt, automatisch übersetzt und durch automatische Tests überprüft. Änderungen, die die Stabilität z. B. durch Seiteneffekte beeinträchtigen könnten, werden somit unmittelbar erkannt und die Zuverlässigkeit und Robustheit der Software verbessert.
<table>
<tr><th>Umsetzungsphase</th><th>Prinzip</th><th>Kriterien</th><th>wirkt gegen</th><th>Querverweis</th></tr>
<tr><td>Beschaffung
Implementierung</td><td>Automatismen
Robustheit</td><td>Automatisierungsgrad
Überwachung
Fehlertolerant
Continuous Integration</td><td>Menschliches Versagen
Fehler in Hard- oder Software</td><td>HVK (Software)</td></tr>
</table></td></tr>
</table>

<table>
<tr><th>Nr.</th><th>Maßnahmen</th></tr>
<tr><td>VM.9.17</td><td>
Einsatz von Diagnose-Software
Während die syntaktische Korrektheit der entwickelten Software durch den Compiler überprüft wird, findet eine semantische Prüfung auf dieser Ebene nicht statt. Mit zusätzlichen automatischen Werkzeugen können jedoch eine Reihe semantischer Fehler entdeckt und das Auftreten von Implementierungsschwächen reduziert werden.
<table>
<tr><th>Umsetzungsphase</th><th>Prinzip</th><th>Kriterien</th><th>wirkt gegen</th><th>Querverweis</th></tr>
<tr><td>Beschaffung
Implementierung</td><td>Automatismen
Fehlertoleranz
Robustheit</td><td>Fehlermeldung
Systemautonomie
Überwachung
Diagnose-Software</td><td>Menschliches Versagen
Fehler in Hard- oder Software</td><td>HVK (Software)</td></tr>
</table>
</td></tr>
<tr><td>VM.9.18</td><td>
Quellcode-Dokumentation
Für die meisten Programmiersprachen existieren Richtlinien zur Dokumentation des Quellcodes. Wird der Code gemäß dieser Richtlinien codiert, können entsprechende Tools automatisch eine ausführliche Zusammenfassung der Implementierung erzeugen, die als Grundlage für die Entwicklung von Test, für die weitere Wartung der Software aber auch zur Beurteilung der Verfügbarkeit verwendet werden kann.
<table>
<tr><th>Umsetzungsphase</th><th>Prinzip</th><th>Kriterien</th><th>wirkt gegen</th><th>Querverweis</th></tr>
<tr><td>Beschaffung
Implementierung</td><td>Transparenz</td><td>Quellcode Dokumentation</td><td>Menschliches Versagen
Fehler in Hard- oder Software</td><td>HVK (Software)</td></tr>
</table>
</td></tr>
</table>

<table>
<tr><th>Nr.</th><th>Maßnahmen</th></tr>
<tr><td>VM.9.19</td><td>
Auswahl und Einsatz von Testverfahren
Testprozesse dienen der Identifizierung von Fehlern in Softwaresystemen und sind daher ein essentieller Baustein zur Realisierung von hochverfügbarer Software. Es ist es generell sinnvoll jeder Phase im Entwicklungsprozess geeignete Tests entgegen zu setzen, um die korrekte Umsetzung der jeweiligen Phase unter Berücksichtigung der jeweiligen Granularität zu überprüfen.
<table>
<tr><th>Umsetzungsphase</th><th>Prinzip</th><th>Kriterien</th><th>wirkt gegen</th><th>Querverweis</th></tr>
<tr><td>Planung und Konzeption
Beschaffung
Implementierung</td><td>Robustheit</td><td>Automatisierungsgrad
Fehlermeldung
Fehlertolerant
Überwachung
Systemautonomie
Unit-Tests</td><td>Menschliches Versagen
Fehler in Hard- oder Software</td><td>HVK (Software)</td></tr>
</table>
</td></tr>
<tr><td>VM.9.20</td><td>
Auswahl der Testkategorie
In Abhängigkeit von den vorliegenden Struktur- und Codeinformationen sollten Blackbox-, Whitebox- oder Greybox-Tests durchgeführt werden um Fehler in der Software zu erkennen und die Stabilität der Implementierung zu verbessern.
<table>
<tr><th>Umsetzungsphase</th><th>Prinzip</th><th>Kriterien</th><th>wirkt gegen</th><th>Querverweis</th></tr>
<tr><td>Planung und Konzeption
Beschaffung
Implementierung</td><td>Fehlertoleranz
Robustheit</td><td>Überwachung
Unit-Tests</td><td>Menschliches Versagen
Fehler in Hard- oder Software</td><td>HVK (Software)</td></tr>
</table>
</td></tr>
</table>

<table>
<tr><th>Nr.</th><th>Maßnahmen</th></tr>
<tr><td>VM.9.21</td><td>Erstellung einer Betriebsdokumentation
Um einen reibungslosen Betrieb zu ermöglichen, muss eine Softwarekomponente in ausreichendem Maße dokumentiert sein. Mit dieser Maßnahme kann der ordnungsgemäße Gebrauch der Software ermöglicht und das Auftreten von Fehlbedienungen reduziert werden.
<table>
<tr><th>Umsetzungsphase</th><th>Prinzip</th><th>Kriterien</th><th>wirkt gegen</th><th>Querverweis</th></tr>
<tr><td>Implementierung
Betrieb
Notfallvorsorge</td><td>Transparenz</td><td>Wartung</td><td>Menschliches Versagen</td><td>HVK (Software)
CobiT, AI4</td></tr>
</table></td></tr>
</table>

Tabelle 1-2: Maßnahmenkatalog Software: Entwicklung und Test

1.3 Implementierung und Betrieb

<table>
<tr><th>Nr.</th><th>Maßnahmen</th></tr>
<tr><td>VM.9.22</td><td>
Software-Virtualisierung
Die Software-Virtualisierung bildet nahezu beliebige Softwareumgebungen nach und stellt diese virtuell zur Verfügung. Im Rahmen der Software-Virtualisierung lassen sich die beiden grundsätzlichen Virtualisierungsebenen Betriebssystem- und Applikationsvirtualisierung im HV-Umfeld umsetzen.
<table>
<tr><th>Umsetzungsphase</th><th>Prinzip</th><th>Kriterien</th><th>wirkt gegen</th><th>Querverweis</th></tr>
<tr><td>Planung und Konzeption
Beschaffung
Implementierung</td><td>Virtualisierung
Skalierbarkeit</td><td>Applikation
Betriebssystem</td><td>Menschliches Versagen
Sabotage, Manipulation
Technische Ermüdung
Fehler in Hard- oder Software
Geplante Ausfallzeiten</td><td>HVK (Prinzipien)</td></tr>
</table>
</td></tr>
<tr><td>VM.9.23</td><td>
Korrekte Installation
Im Rahmen von Softwareinstallationen im HV-Umfeld sollten nur die tatsächlich benötigten Funktionen installiert werden, um die Wahrscheinlichkeit von Angriffpunkten oder fehlerhaften Modulen zu minimieren. Auf der Betriebssystemebene wird mit einer so genannten Härtung des System erreicht, dass nur benötigte Bibliotheken und Treiber vorhanden sind. Insgesamt kann so eine größere Zuverlässigkeit und Robustheit des Systems erreicht werden.
<table>
<tr><th>Umsetzungsphase</th><th>Prinzip</th><th>Kriterien</th><th>wirkt gegen</th><th>Querverweis</th></tr>
<tr><td>Planung und Konzeption
Implementierung</td><td>Separation
Priorisierung
Robustheit</td><td>Tests
Anweisungen
Reifegrad</td><td>Menschliches Versagen
Fehler in Hard- oder Software
Sabotage, Manipulation</td><td>HVK (Software)
CobiT, AI7</td></tr>
</table>
</td></tr>
</table>

<table>
<tr><th>Nr.</th><th>Maßnahmen</th></tr>
<tr><td>VM.9.24</td><td>
Qualität der Eingabedaten
Fehlerhafte oder bewusst manipulierte Eingabedaten führen zu einer Beeinträchtigung der Verfügbarkeit, Organisatorische oder technische Maßnahmen zur Sicherung der Qualität der Eingabedaten steigern die Verfügbarkeit von Softwaresystemen und führen zu einer Verbesserung der Fehlertoleranz der Software.
<table>
<tr><th>Umsetzungsphase</th><th>Prinzip</th><th>Kriterien</th><th>wirkt gegen</th><th>Querverweis</th></tr>
<tr><td>Planung und Konzeption
Implementierung
Betrieb</td><td>Automatismen
Fehlertoleranz
Robustheit</td><td>Fehlertolerant
Plausibilität
Fehlbedienungstolerant</td><td>Menschliches Versagen
Sabotage, Manipulation</td><td>HVK (Software)</td></tr>
</table>
</td></tr>
<tr><td>VM.9.25</td><td>
Korrekte Konfiguration
Eine fehlerhafte Konfiguration kann die Verfügbarkeit einiger oder aller Funktionen des Systems gefährden. Komplexe Systemkonfigurationen erhöhen das Risiko von Fehlern bei Änderungen der Einstellungen. Daher sind in jedem Fall technische und organisatorische Maßnahmen zur Sicherstellung einer korrekten Systemkonfiguration zu realisieren.
<table>
<tr><th>Umsetzungsphase</th><th>Prinzip</th><th>Kriterien</th><th>wirkt gegen</th><th>Querverweis</th></tr>
<tr><td>Planung und Konzeption
Implementierung
Betrieb</td><td>Fehlertoleranz
Robustheit</td><td>Tests
Anweisungen</td><td>Menschliches Versagen</td><td>HVK (Software)</td></tr>
</table>
</td></tr>
</table>

<table>
<tr><th>Nr.</th><th>Maßnahmen</th></tr>
<tr><td>VM.9.26</td><td>
Einsatz wieder verwendbarer Komponenten
Ähnlich dem Design-Pattern Ansatz beim Software-Design sollten auch bei der Implementierung bestehende Module, deren hohe Verfügbarkeit bereits nachgewiesen wurde, verwendet werden. Vorrausetzung für die Verwendbarkeit bestehender Module ist eine ausführliche und exakte Dokumentation der Schnittstellen und Funktionen der eingesetzten Module.
<table>
<tr><th>Umsetzungsphase</th><th>Prinzip</th><th>Kriterien</th><th>wirkt gegen</th><th>Querverweis</th></tr>
<tr><td>Beschaffung
Implementierung
Betrieb</td><td>Robustheit</td><td>Modularisierung
Wiederverwendbare Komponente</td><td>Menschliches Versagen
Fehler in Hard- oder Software</td><td>HVK (Software)</td></tr>
</table>
</td></tr>
<tr><td>VM.9.27</td><td>
Aktive Fehlerbehandlung
Jede Funktion muss so implementiert werden, dass Fehlerzustände explizit erwartet und aktiv behandelt werden. Dies gilt insbesondere auch für die Kommunikation mit anderen Funktionen und Modulen, bei denen Ergebnisse soweit möglich auf Sinnhaltigkeit überprüft werden müssen. Dies bildet die Voraussetzung für die Anwendung von Folgemaßnahmen wie Fehlermaskierung, Fehlerkompensation oder Fehlerbehebung.
<table>
<tr><th>Umsetzungsphase</th><th>Prinzip</th><th>Kriterien</th><th>wirkt gegen</th><th>Querverweis</th></tr>
<tr><td>Beschaffung
Implementierung
Betrieb</td><td>Fehlertoleranz
Automatismen
Robustheit
Autonomie</td><td>Systemautonomie
Automatisierungsgrad
Überwachung
Fehlermeldung
Fehlertolerant
Fehlbedienungstolerant
Wiederherstellung
Aktive Fehlerbehandlung</td><td>Menschliches Versagen
Fehler in Hard- oder Software
Sabotage, Manipulation</td><td>HVK (Software)</td></tr>
</table>
</td></tr>
</table>

<table>
<tr><th>Nr.</th><th>Maßnahmen</th></tr>
<tr><td>VM.9.28</td><td>
Protokollierung von Konfigurationsänderungen
Jegliche Änderungen an der Konfiguration einer Softwarekomponente bzw. deren Umgebung sollte protokolliert werden, so dass eine spätere Nachvollziehbarkeit der Änderungen im Fehlerfall oder zur Optimierung der Konfiguration möglich ist.
<table>
<tr><th>Umsetzungsphase</th><th>Prinzip</th><th>Kriterien</th><th>wirkt gegen</th><th>Querverweis</th></tr>
<tr><td>Betrieb</td><td>Transparenz</td><td>Protokollierungsgrad</td><td>Menschliches Versagen
Fehler in Hard- oder Software
Sabotage, Manipulation</td><td>HVK (Software)</td></tr>
</table>
</td></tr>
<tr><td>VM.9.29</td><td>
Schulung des Personals
Neben der Bereitstellung von Dokumentation muss das Personal in der korrekten Verwendung der Software eingewiesen und geschult werden. Dies gilt sowohl für Endbenutzer von Softwarekomponenten (Anwender) als auch für Administratoren.
<table>
<tr><th>Umsetzungsphase</th><th>Prinzip</th><th>Kriterien</th><th>wirkt gegen</th><th>Querverweis</th></tr>
<tr><td>Beschaffung
Implementierung
Beschaffung
Notfallvorsorge</td><td>Transparenz
Fehlertoleranz</td><td>Stand der Schulungsteilnahme
Anzahl Anwenderschulungen</td><td>Menschliches Versagen</td><td>HVK (Software) VM.2.6
CobiT, AI4
CobiT, PO7</td></tr>
</table>
</td></tr>
</table>

<table>
<tr><th>Nr.</th><th>Maßnahmen</th></tr>
<tr><td>VM.9.30</td><td>
Schutz vor Manipulation
Die Verfügbarkeit von Software wird maßgeblich davon beeinflusst welche Möglichkeiten einer beabsichtigten und unbeabsichtigten Manipulation der Software existieren. Es sollten Maßnahmen umgesetzt werden, um die Möglichkeiten eines Angreifers zur bewussten Manipulation beziehungsweise von unbeabsichtigten Änderungen einzuschränken.
<table>
<tr><th>Umsetzungsphase</th><th>Prinzip</th><th>Kriterien</th><th>wirkt gegen</th><th>Querverweis</th></tr>
<tr><td>Beschaffung
Implementierung
Betrieb</td><td>Automatismen
Fehlertoleranz
Robustheit</td><td>Basis-Sicherheitscheck</td><td>Sabotage, Manipulation</td><td>HVK (Software)
BSI 100-2</td></tr>
</table>
</td></tr>
<tr><td>VM.9.31</td><td>
Abschalten nicht benötigter Leistungsmerkmale
Modulare Softwarekomponenten, die eine An- und Abschaltung verschiedener Leistungsmerkmale anbieten, sollten derart konfiguriert werden, dass nicht mehr als die für den Betrieb erforderliche Funktionalität angeboten wird. Dies verringert die Komplexität der Komponente und damit deren Fehleranfälligkeit sowie die Anzahl an Angriffsmöglichkeiten.
<table>
<tr><th>Umsetzungsphase</th><th>Prinzip</th><th>Kriterien</th><th>wirkt gegen</th><th>Querverweis</th></tr>
<tr><td>Implementierung
Betrieb</td><td>Fehlertoleranz
Separation</td><td>Fehlertolerant
Fehlbedienungstolerant</td><td>Menschliches Versagen
Fehler in Hard- oder Software
Sabotage, Manipulation</td><td>HVK (Software)</td></tr>
</table>
</td></tr>
</table>

<table>
<tr><th>Nr.</th><th>Maßnahmen</th></tr>
<tr><td>VM.9.32</td><td>
Sichere Grundkonfiguration
Falls eine Softwarekomponente durch verschiedene Anwendungsparameter beeinflusst wird, sollte diese im Auslieferungszustand mit einer Konfiguration versehen werden, die ein höchstmögliches Maß an Zuverlässigkeit und Stabilität bietet.
<table>
<tr><th>Umsetzungsphase</th><th>Prinzip</th><th>Kriterien</th><th>wirkt gegen</th><th>Querverweis</th></tr>
<tr><td>Implementierung
Betrieb</td><td>Fehlertoleranz
Robustheit</td><td>Fehlertolerant</td><td>Menschliches Versagen
Fehler in Hard- oder Software
Sabotage, Manipulation</td><td>HVK (Software)
GSK M 4.237</td></tr>
</table>
</td></tr>
<tr><td>VM.9.33</td><td>
Ein-Dienst-pro-Host-System
Werden verschiedene Softwarekomponenten auf demselben Host-System betrieben, besteht die Möglichkeit der indirekten gegenseitigen Beeinflussung. Aufgrund der Fehleranfälligkeit von Softwarekomponenten im Allgemeinen besteht daher die Möglichkeit, dass Fehlersituationen eine Auswirkung auf andere Dienste desselben Host-Systems haben, bis hin zum Totalausfall anderer Dienste. Daher sollte bei der Installation von Software auf Host-Systemen das Ein-Dienst-pro-Host-Prinzip umgesetzt und durch geeignete Separation eine Verbesserung der Robustheit der Gesamtarchitektur sowie der Skalierbarkeit erreicht werden.
<table>
<tr><th>Umsetzungsphase</th><th>Prinzip</th><th>Kriterien</th><th>wirkt gegen</th><th>Querverweis</th></tr>
<tr><td>Planung und Konzeption
Betrieb
Notfallvorsorge</td><td>Fehlertoleranz
Robustheit
Separation
Skalierbarkeit</td><td>Fehlertolerant
Physikalisch/technisch
Modularisierung</td><td>Menschliches Versagen
Fehler in Hard- oder Software
Sabotage, Manipulation</td><td>HVK (Software)</td></tr>
</table>
</td></tr>
</table>

Nr.	*Maßnahmen*				
VM.9.34	**Sicherstellung der Integrität und Authentizität** Da in der Softwareentwicklung der Einsatz wieder verwendbarer Software angestrebt wird, besitzen Softwarekomponenten in der Regel zahlreiche Abhängigkeiten zu anderen Softwarepaketen. Um einen ordnungsgemäßen Betrieb gewährleisten zu können, ist vor der Inbetriebnahme der Softwarekomponente zu prüfen, ob die Integrität und Authentizität der Softwarepakete gewährleistet sind. Hierzu können verschiedene Mechanismen wie Prüfsummen oder digitale Signaturen für Softwarepakete eingesetzt werden.				
	Umsetzungsphase	**Prinzip**	**Kriterien**	**wirkt gegen**	**Querverweis**
	Betrieb	Redundanz	Information Basis-Sicherheitscheck	Menschliches Versagen Fehler in Hard- oder Software Sabotage, Manipulation	HVK (Software) BSI 100-2
VM.9.35	**Einspielen von Patches und Upgrades** Sobald Schwachstellen bekannt sind, die durch das Einspielen eines Patches beseitigt werden können, sollte ein Update von Software bzw. abhängigen Softwarepaketen durchgeführt werden um die Stabilität und Zuverlässigkeit der Software während des Betriebs zu gewährleisten.				
	Umsetzungsphase	**Prinzip**	**Kriterien**	**wirkt gegen**	**Querverweis**
	Betrieb	Robustheit Fehlertoleranz	Wartung Basis-Sicherheitscheck	Menschliches Versagen Fehler in Hard- oder Software Sabotage, Manipulation	HVK (Software) BSI 100-2

<table>
<tr><th>Nr.</th><th>Maßnahmen</th></tr>
<tr><td>VM.9.36</td><td>
Einrichtung eines Referenzsystems
Nicht nur bei komplexen Anwendungen sollte ein, zu einem produktiv eingesetzten System weitestgehend identisches Referenzsystem eingerichtet werden. Dies ermöglicht die Nachverfolgung von Fehlersituation oder die Durchführung von Schulungsübungen ohne Beeinflussung des produktiven Betriebs. Weiterhin besteht die Möglichkeit, dieses System in den Testprozess für Updates und Patches einzubeziehen.
<table>
<tr><th>Umsetzungsphase</th><th>Prinzip</th><th>Kriterien</th><th>wirkt gegen</th><th>Querverweis</th></tr>
<tr><td>Planung und Konzeption
Beschaffung
Implementierung
Betrieb
Notfallvorsorge</td><td>Redundanz
Robustheit</td><td>Physikalisch/technisch
Organisatorisch
Wiederherstellung
Modularisierung</td><td>Menschliches Versagen
Fehler in Hard- oder Software
Sabotage, Manipulation</td><td>HVK (Software)</td></tr>
</table>
</td></tr>
</table>

Tabelle 1-3: Maßnahmenkatalog Software: Implementierung und Betrieb

1.4 Notfallvorsorge und Behandlung

<table>
<tr><th>Nr.</th><th>Maßnahmen</th></tr>
<tr><td>VM.9.37</td><td>
Mehrfachprogrammierung
Im HV-Umfeld sollte zur Vorsorge für Notfälle sowie im Rahmen der Notfallbehandlung Software eingesetzt werden, die sich aufgrund von Mehrfachprogrammierung Fehler erkennend (n+2) oder Fehler korrigierend (n+3) verhält. Alternativ sowie ergänzend kann auch der Einsatz funktional identischer Software unterschiedlicher Hersteller erfolgen.
<table>
<tr><th>Umsetzungsphase</th><th>Prinzip</th><th>Kriterien</th><th>wirkt gegen</th><th>Querverweis</th></tr>
<tr><td>Planung und Konzeption
Beschaffung
Notfallvorsorge</td><td>Redundanz</td><td>SpoF
Grad
Funktionell
Information
Diversität
Fehlertolerant</td><td>Technische Ermüdung
Fehler in Hard- oder Software</td><td>HVK (Software)
HVK (Prinzipien)</td></tr>
</table>
</td></tr>
</table>

<table>
<tr><th>Nr.</th><th>Maßnahmen</th></tr>
<tr><td>VM.9.38</td><td>
Einsatz fehlertoleranter Software
Im Umfeld von HV-Architekturen sollen grundsätzlich Softwarekomponenten eingesetzt werden, die eine robuste Implementierung aufweisen, eine Maskierung fehlerhafter Ereignisse durchführen und automatische Fehlerkorrekturverfahren oder Plausibilitätsprüfungen zur Neutralisierung von Beeinträchtigungen einsetzen.
<table>
<tr><th>Umsetzungsphase</th><th>Prinzip</th><th>Kriterien</th><th>wirkt gegen</th><th>Querverweis</th></tr>
<tr><td>Planung und Konzeption
Beschaffung
Betrieb
Notfallvorsorge</td><td>Fehlertoleranz
Redundanz</td><td>Fehlertolerant
Automatisierungsgrad
Überwachung</td><td>Technische Ermüdung
Fehler in Hard- oder Software</td><td>HVK (Prinzipien)</td></tr>
</table>
</td></tr>
<tr><td>VM.9.39</td><td>
Einsatz definierter Testprozesse
Zur Sicherstellung der Eignung für den Einsatz in HV-Architekturen sind hinreichend ausführliche und umfangreiche Tests im Rahmen von definierten Testprozessen in den Entwicklungsprozess von Software zu integrieren, um die Gefahr von Ausfällen aufgrund von Softwarefehlern zu reduzieren. Der Testprozess muss auch in die Wartungs- und Pflegephase der Software eingebunden werden, um die HV-Eigenschaften während des gesamten Lebenszyklus zu gewährleisten.
<table>
<tr><th>Umsetzungsphase</th><th>Prinzip</th><th>Kriterien</th><th>wirkt gegen</th><th>Querverweis</th></tr>
<tr><td>Planung und Konzeption
Beschaffung
Implementierung
Betrieb
Notfallvorsorge</td><td>Automatismen
Autonomie
Fehlertoleranz
Robustheit</td><td>Überwachung</td><td>Menschliches Versagen
Fehler in Hard- oder Software</td><td>HVK (Software)</td></tr>
</table>
</td></tr>
</table>

Nr.	*Maßnahmen*				
VM.9.40	**Errichtung einer Cluster-fähigen Firewall** Eine Firewall muss umfangreiche Statusinformationen sammeln, die sie für ihre Funktionsweise benötigt. Die innerhalb der HV-Architektur eingesetzte Firewall muss cluster-fähig sein, d. h. sie muss Status- und Zustandsinformationen mit den weiteren Firewalls im Cluster austauschen können. Im Fail-over muss eine andere Firewall im Cluster die Steuerung der Sitzung mit einer für den Benutzer hinreichenden Sitzungstransparenz innerhalb kürzester Zeit übernehmen.				
	Umsetzungsphase	**Prinzip**	**Kriterien**	**wirkt gegen**	**Querverweis**
	Planung und Konzeption Beschaffung	Redundanz	Aktivierungszeit	Fehler in Hard- oder Software	HVK (Netzwerk)
VM.9.41	**Redundanz statischer Schlüssel im Gateway-Cluster** Statische Schlüssel (Preshared Keys, PSK) müssen optimalerweise schon a priori auf die Stand-By-Systeme aufgebracht werden, damit diese bei einem Fail-Over unmittelbar eingesetzt werden können.				
	Umsetzungsphase	**Prinzip**	**Kriterien**	**wirkt gegen**	**Querverweis**
	Planung und Konzeption Implementierung	Redundanz	Aktivierungszeit	Fehler in Hard- oder Software	HVK (Netzwerk)

<table>
<tr><th>Nr.</th><th>Maßnahmen</th></tr>
<tr><td>VM.9.42</td><td>
Einsatz eines DNS Baum-Cluster
Bei DNS-Servern sollte das Konzept des „Versteckten Primärservers“ (Hidden primary) verwendet werden. Von dem Primär-Server erfolgt hierbei ein Zonentransfer auf alle Sekundär-Server, die so alle auf dem gleichen Stand gehalten werden und die Client-Anfragen bearbeiten. Es handelt sich dabei um eine Ausprägung eines Baum-Clusters, bei dem jedoch auf den Wurzel-Knoten nicht zugegriffen wird.
<table>
<tr><th>Umsetzungsphase</th><th>Prinzip</th><th>Kriterien</th><th>wirkt gegen</th><th>Querverweis</th></tr>
<tr><td>Planung und Konzeption
Beschaffung
Implementierung</td><td>Redundanz</td><td>SpoF</td><td>Fehler in Hard- oder Software</td><td>HVK (Netzwerk)</td></tr>
</table>
</td></tr>
<tr><td>VM.9.43</td><td>
Protokollierung von Systemereignissen
Die Protokollierung von Systemereignissen (typischerweise in Logdateien) bietet verschiedene Möglichkeiten zur Untersuchung der Softwareverfügbarkeit. Fehlerzustände der Softwarekomponente sowie Informationen darüber, welche Folge von Systemereignissen zu einer Fehlersituation führen, sollten festgehalten werden. Die Protokollierungsfunktionalität einer Softwarekomponente muss bereits in der Entwicklungsphase berücksichtigt und implementiert werden.
<table>
<tr><th>Umsetzungsphase</th><th>Prinzip</th><th>Kriterien</th><th>wirkt gegen</th><th>Querverweis</th></tr>
<tr><td>Beschaffung
Implementierung
Betrieb
Notfallvorsorge</td><td>Automatismen</td><td>Fehlermeldung
Überwachung</td><td>Menschliches Versagen
Fehler in Hard- oder Software</td><td>HVK (Software)</td></tr>
</table>
</td></tr>
</table>

<table>
<tr><th>Nr.</th><th>Maßnahmen</th></tr>
<tr><td>VM.9.44</td><td>
Fehlerreports von Softwarekomponenten
In Fehlersituationen können Informationen zur Fehlerdiagnose auf Nachfrage hin an den Hersteller einer Softwarekomponente gesendet werden. Dies kann insbesondere für Standardsoftware, die in zahlreichen unterschiedlichen Umgebungen eingesetzt wird dazu beitragen, die Robustheit der Anwendung gegenüber fehlerhaften Eingaben beziehungsweise fehlerhafter Bedienung zu optimieren.
<table>
<tr><th>Umsetzungsphase</th><th>Prinzip</th><th>Kriterien</th><th>wirkt gegen</th><th>Querverweis</th></tr>
<tr><td>Betrieb
Notfallvorsorge</td><td>Automatismen
Fehlertoleranz</td><td>Fehlermeldung
Automatisierungsgrad
Überwachung
Reaktionszeit
Systemautonomie</td><td>Menschliches Versagen
Fehler in Hard- oder Software
Sabotage, Manipulation</td><td>HVK (Software)</td></tr>
</table>
</td></tr>
<tr><td>VM.9.45</td><td>
Erstellung eines Notfallhandbuches
Um eine schnelle Reaktion auf Notfallereignisse zu ermöglichen, sollten auch – soweit möglich – die Behebung von Teil- bzw. Totalausfällen von Softwarekomponenten im Notfallhandbuch bedacht und die erforderlichen Maßnahmen für ein derartiges Notfallereignis definiert und beschrieben werden.
<table>
<tr><th>Umsetzungsphase</th><th>Prinzip</th><th>Kriterien</th><th>wirkt gegen</th><th>Querverweis</th></tr>
<tr><td>Planung und Konzeption
Betrieb
Notfallvorsorge</td><td>Robustheit</td><td>Wiederherstellung
Aktivierungszeit
Aktivierung
Reaktionszeit
Systemautonomie
Reifegrad</td><td>Menschliches Versagen
Fehler in Hard- oder Software
Sabotage, Manipulation</td><td>HVK (Software)
BSI 100-4
CobiT, DS4</td></tr>
</table>
</td></tr>
</table>

<table>
<tr><th>Nr.</th><th>Maßnahmen</th></tr>
<tr><td>VM.9.46</td><td>Erstellung eines Wideranlaufplans
Der Wiederanlaufprozess bei Ausfall einer Softwarekomponente soll in Form eines Wiederanlaufplans dokumentiert werden. Hierbei müssen insbesondere auch abhängige Softwarekomponenten berücksichtigt werden.
<table>
<tr><th>Umsetzungsphase</th><th>Prinzip</th><th>Kriterien</th><th>wirkt gegen</th><th>Querverweis</th></tr>
<tr><td>Notfallvorsorge
Planung und Konzeption
Betrieb</td><td>Robustheit
Priorisierung</td><td>Wiederherstellung
Aktivierungszeit
Reifegrad</td><td>Menschliches Versagen
Fehler in Hard- oder Software
Sabotage, Manipulation</td><td>HVK (Software)
BSI 100-4
CobiT, DS4</td></tr>
</table></td></tr>
<tr><td>VM.9.47</td><td>Automatisierung des Wiederanlaufs
Soweit der Ausfall einer Softwarekomponente durch einen vorübergehenden Fehler hervorgerufen wurde (beispielsweise durch eine fehlerhafte Reaktion auf eine äußere Eingabe), sollte durch die Automatisierung des Wiederanlaufs die Ausfalldauer der Softwarekomponente minimiert werden.
<table>
<tr><th>Umsetzungsphase</th><th>Prinzip</th><th>Kriterien</th><th>wirkt gegen</th><th>Querverweis</th></tr>
<tr><td>Betrieb
Notfallvorsorge</td><td>Automatismen
Robustheit
Fehlertoleranz</td><td>Neustarts
Reaktionszeit
Wiederherstellung
Automatisierungsgrad
Systemautonomie
Automatisierung des Wiederanlaufs
Reifegrad</td><td>Menschliches Versagen
Fehler in Hard- oder Software
Sabotage, Manipulation</td><td>HVK (Software)
BSI 100-4
CobiT, DS4</td></tr>
</table></td></tr>
</table>

Tabelle 1-4: Maßnahmenkatalog Software: Notfallvorsorge und Behandlung