RICHTLINIE SOFTWAREENTWICKLUNG

Datum:	09.08.2022	
Version:	V003	
Autor*in:	Eike Schlingensief Paul Bottin	VDI GmbH - IT-IS VDI GmbH - IT-AE
Eigentümer*in:	Leonard Sadrinna	VDI GmbH - IT-AE
Status:	Freigegeben	

Historie

Version	Datum	Änderung	Autor
SADM001	14.10.2014	Ersterstellung	Schlingensief
SADM002	12.01.2015	Begrenzung des Geltungsbereiches der Richtlinie	Schlingensief
SADM003	13.01.2016	Funktionstrennung aufgenommen	Schlingensief
V001	15.01.2016	Übernahme in die IT-Dokumentation mit neuer Formatvorlage.	Holzapfel
V002	03.08.2022	Harmonisierung mit IT-AE Entwicklungssystem Hinzufügen der Punkte • Änderung Entwurfsphase in PM • Allgemeine Grundsätze • Systembild • Paketmanager • Container • Penetrationstests • Deployment • Server Dokumentation • Umgebungstrennung Genderneutrale Sprache und Formulierungen angepasst	Bottin
V003	08.08.2022	Übersetzung nach Englisch Anpassung an neue Corporate Identity	Bottin
V003	09.08.2022	Freigabe – Gültig ab sofort	Sadrinna

Inhaltsverzeichnis

Historie II

Inhaltsverzeichnis III

1 Glossar 1

2 Einleitung 2

3 Geltungsbereich 3

4 Projektmanagement 4

4.1 Projektstrukturplan 4

4.2 Geltungsbereich und Liefergegenstände 4

4.3 Projektsteckbrief 4

4.4 Risikoanalyse 5

4.5 Statusbericht 6

5 Entwicklung 7

5.1 Allgemeine Grundsätze für die Softwareentwicklung 7

5.2 Systembild / Software-Engineering 7

5.3 Programmiersprache 8

5.4 Frameworks 8

5.5 Paketmanager 9

5.6 Container 9

5.7 Versionierung 9

5.8 Tests 9

5.8.1 Funktionstests (Unit-Tests) 10

5.8.2 Verhaltenstests 10

5.8.3 Penetrationstests 10

5.9 Reviews 10

5.10 Validierung 11

5.11 Aktualisierung 11

5.12 Berechtigungskonzept 11

5.13 Authentifizierung und Passwort Management 11

5.14 Session Management 13

5.15 Zugriffskontrolle 13

5.16 Fehlerbehandlung 14

5.17 Logging 14

5.17.1 Sicherheitsverstöße 14

5.17.2 Personenbezogene Daten 14

5.18 Datenintegrität 14

5.19 Datenbanken 14

5.20 Passwörter 14

5.21 Systemkonfiguration 14

5.22 Dateiuploads 15

5.23 Speicher- bzw. Ressourcenverwaltung 15

5.24 Programmierkonventionen (Coding Guidelines) 16

5.25 Kryptographie und Kommunikation 16

6 Deployment 17

7 Dokumentation 18

7.1 Benutzerhandbuch 18

7.2 Technische Dokumentation 18

7.3 Server Dokumentation 18

8 Sicherheit der Entwicklungsumgebung 19

8.1 Backup 19

8.2 Zugriff 19

8.3 Testdaten 19

8.4 Funktionstrennung 19

8.5 Entwicklungs- und Testumgebung (Umgebungstrennung) 19

9 Entwickelnde 21

10 Bezugsdokumente 22

1 Glossar

Kann-Kriterien

Formulierungen im Konjunktiv wie z.B. Soll, sollte, müsste, kann - sind optionale Anforderungen.

Muss-Kriterien

Formulierungen im Imperativ wie z.B. Muss, darf nicht, ist - sind obligatorische Anforderungen.

VCS / GIT

Versionskontrollsystem (engl. Version Control System) – ist eine Einrichtung zur Änderungsverfolgung von Quellcode. In der Anwendungsentwicklung der VDI Gruppe wird bevorzugt das weitverbreitete GIT-System verwendet.

Deployment

Der Vorgang des Ausrollens einer Anwendung bzw. Funktion für die Hauptnutzenden.

Geschäftsprozessmodellierung / Business process modeling (BPM, BPM Diagram)

Diagrammtyp zur Modellierung von Abläufen und Prozessen

Bei der Geschäftsprozessmodellierung (engl.: Business Process Modeling) werden Geschäftsprozesse oder Ausschnitte daraus abstrahiert, meist grafisch dargestellt und somit modelliert. Der Schwerpunkt liegt auf dem Darstellen des Ablaufs, aber auch Daten und Organisation (bzw. Organisationseinheiten) können modelliert werden. Geschäftsprozessmodellierung ist ein zentraler Aspekt der ganzheitlichen Unternehmensabbildung und wird in der Regel als ein Teil des Geschäftsprozessmanagements verstanden. (aus Wikipedia)

Entity-Relationship-Modell (ER Diagram)

Diagrammtyp zur Modellierung von Datenstrukturen

Das Entity-Relationship-Modell – kurz ER-Modell oder ERM; deutsch so viel wie: Modell (zur Darstellung) von Dingen, Gegenständen, Objekten (= ‚entities') und der Beziehungen/Zusammenhänge zwischen diesen (= ‚relationship') – dient dazu, im Rahmen der semantischen Datenmodellierung den in einem gegebenen Kontext (z. B. einem Projekt zur Erstellung eines Informationssystems) relevanten Ausschnitt der realen Welt zu bestimmen und darzustellen. Das ER-Modell besteht im Wesentlichen aus einer Grafik (ER-Diagramm, Abk. ERD) sowie einer Beschreibung der darin verwendeten Elemente.

Ein ER-Modell dient sowohl in der konzeptionellen Phase der Anwendungsentwicklung der Verständigung zwischen Anwendern und Entwicklern (dabei wird nur das Was behandelt, d. h. fachlich-sachliche Gegebenheiten, nicht das Wie, z. B. die Technik) als auch in der Implementierungsphase als Grundlage für das Design der – meist relationalen – Datenbank. (aus Wikipedia)

2 Einleitung

Diese Richtlinie beschreibt die Anforderungen an den Entwicklungsprozess einer Software von der Entwurfsphase bis zum laufenden Betrieb. Dies dient unter anderem dazu, die relevanten Punkte der Informationssicherheit mit einzubeziehen. Weiterhin werden generelle Anforderungen an das Projektmanagement sowie der Dokumentation formuliert, die dem reibungslosen Entwicklungs- bzw. Weiterentwicklungsprozess dienen.

In der Richtlinie wird unterschieden, ob es sich bei der Anwendung um ein Arbeitsmittel oder um ein reines Informationsboard handelt. Ein Arbeitsmittel, das sensible Informationen verarbeitet, muss höheren Sicherheitsstandards gerecht werden als ein reines Informationsboard mit öffentlichen Informationen. Aus diesem Grund müssen bei einer Anwendung, die sensible Informationen verarbeitet, alle in dieser Richtlinie gestellten Anforderungen erfüllt sein, auch wenn sie als optional gekennzeichnet sind. Für alle anderen Anwendungen wie z.B. Informationsboards müssen optionale Anforderungen nicht berücksichtigt werden.

3 Geltungsbereich

Diese Richtlinie gilt für Mitarbeitende der VDI Gruppe sowie für externe Dienstleistende, die im Rahmen ihrer Tätigkeit für die VDI Gruppe Software oder Skripte entwickeln.

4 Projektmanagement

Sollte keine Richtline bezüglich Projektmanagement vorliegen gelten folgende Minimalanforderungen:

4.1 Projektstrukturplan

Vor jedem neuen Softwareentwicklungsprojekt muss ein Projektstrukturplan bzw. Anforderungskatalog erstellt werden, um bei Projektabnahme sicherzustellen, dass am Ende alle Funktionen korrekt umgesetzt wurden und um zu prüfen, ob der Arbeitsaufwand richtig abgeschätzt wurde. Das gleiche gilt bei Aktualisierungen einzelner Komponenten oder einer Neuentwicklung der Anwendung. Der Projektstrukturplan soll je nach Größe des Projekts in Teilprojekte, Meilensteine bzw. Komponenten gegliedert sein und jeweils Arbeitspakete definieren, die dann für eine Ressourcenplanung und für die Fortschrittsüberprüfung herangezogen werden können.

Bei agilen Projekten kann ein Anforderungskatalog auch über Einzelanforderungen in Jira abgebildet werden. Wichtig ist hierbei, dass jede Einzelanforderung über Akzeptanzkriterien verfügt, welche für die Abnahme bindend sind.

4.2 Geltungsbereich und Liefergegenstände

Es ist darauf zu achten, den Geltungsbereich der Anwendung durch die Formulierung von Anwendungsfällen einzugrenzen und Limits bzw. Grenzen der Anwendung zu definieren. Dabei werden Formulierungen wie im Glossar dieser Richtlinie verwendet, um optionale und obligatorische Anforderungen zu kennzeichnen. Der Detailgrad der Anforderungen muss hinreichend genau sein, um diese umsetzen zu können. Verwendete Abkürzungen bzw. Akronyme sind, sofern sie nicht allgemein bekannt sind, in einem Glossar aufzuführen.

Weiterhin sind die Schnittstellen zwischen Komponenten der Anwendung sowie ggf. zu Benutzenden und externen Systemen zu bestimmen und zu benennen. Wenn möglich ist auf entsprechende Dokumentation zu verweisen.

Für umfangreiche Benutzerschnittstellen sind vorab Skizzen bzw. Wireframes anzufertigen und mit den Stakeholdern abzustimmen.

4.3 Projektsteckbrief

Jedes Projekt muss über ein Datenblatt mit den wichtigsten Informationen für die Installation und Konfiguration der Anwendung verfügen. Das kann z.B. bei Webanwendungen als bevorzugt englische Readme-Datei im Projektstammverzeichnis geschehen. Bei integrierten Anwendungen – wie z.B. dem Dokumentenmanagementsystem Doxis4 DMS – kann der Projektsteckbrief an einer zentralen Stelle – wie z.B. einer SharePoint-Seite oder in Confluence – angelegt werden. Dies hat den Hintergrund, dass die Einarbeitungszeit bei Mitarbeitenden, die oft in sehr viele Projekte involviert sind, verringert werden soll.

Der Projektsteckbrief muss mindestens die folgenden Informationen enthalten:

- Kurzbeschreibung des Projekts

    - 3-5 Zeilen möglichst in einfacher Sprache, so dass alle Stakeholder verstehen, wo die Schwerpunkte des Projekts liegen.
- Ausgangssituation (Ist-Zustand)
    - Welche Werkzeuge wurden bisher eingesetzt?
    - Welche Hürden gibt es mit der aktuellen Situation?
- Ziel (Soll-Zustand)
    - die konkreten wesentlichen Ziele des Projekts
    - Nutzenversprechen des Projekts
- Projektabgrenzung/ Geltungsbereich
    - Grenzfälle möglichst genau formulieren
    - Eingrenzung auf z.B. Regionen, Themen bzw. Produkte
    - Ggf. Formulieren, was nicht mehr im Geltungsbereich der Anwendung liegt
- Stakeholder
    - Auftraggebende und -nehmende
    - Entscheidungstragende benennen
    - Projektbeteiligte
    - Betroffene
    - Ggf. Nutzergruppen benennen
- Bestandteile der Anwendung
- Schnittstellen zu anderen Systemen
    - System und Prozesse
- Benötigte Ressourcen und Systemvoraussetzungen
    - Freigegebenes Budget
- Risiken
- Schätzung zum Aufwand
- Timeline
    - Deadline

Auf bestehende Projekte sollte im Rahmen der Anforderungsformulierung nur verwiesen werden, wenn für die Projekte auch die Erfordernisse dieser Richtlinie eingehalten wurden. D.h. Wenn auf Funktionen in anderen Projekten verwiesen wird, sollten diese Projekte entsprechend dieser Richtlinie dokumentiert sein.

Das heißt, dass unter Umständen bei der Weiterentwicklung von alten Projekten erhöhter Aufwand entsteht, der ausreichend zu berücksichtigen ist.

Der Projektsteckbrief ist über die Dauer der Entwicklung in regelmäßigen Abständen auf Aktualität zu prüfen und ggf. zu aktualisieren.

4.4 Risikoanalyse

Risiken müssen bei der Erstellung des Projektsteckbriefs betrachtet und entsprechend im Projektsteckbrief verschriftlicht werden. Bei jeder Veränderung (Change) muss eine erneute Risikoanalyse durchgeführt werden.

Bei der Risikoanalyse werden Szenarien gesammelt, die den Projekterfolg gefährden. Für jedes Risiko wird die Eintrittswahrscheinlichkeit, die Auswirkung auf den Projekterfolg und mögliche Gegenmaßnahmen ermittelt.

4.5 Statusbericht

In einem Statusbericht wird der Projektfortschritt in regelmäßigen Abständen dokumentiert. Er besteht i.d.R. aus einer Seite und enthält folgende Informationen über das Projekt:

- planmäßige Fertigstellung – z.B. als Ampeldarstellung
- planmäßige Budgetierung – z.B. als Ampeldarstellung
- werden die qualitativen Anforderungen erfüllt – z.B. als Ampeldarstellung
- aktueller Fertigstellungsgrad – in Prozent [%]
- ggf. Risiken
- Aktuelles wie z.B. Entscheidungsbedarfe oder nächste Schritte

5 Entwicklung

5.1 Allgemeine Grundsätze für die Softwareentwicklung

ZWÖLF-FAKTOREN-APP – für Webanwendung sind die Prinzipien der 12-Faktoren-App ausreichend zu berücksichtigen.

https://12factor.net/de/

KEEP IT SIMPLE (KISS-PRINZIP) – wenn es mehrere Lösungsmöglichkeiten gibt, ist diejenige auszuwählen, die mit der geringsten Komplexität auskommt.

DON'T REPEAT YOURSELF – ähnliche Anforderungen bzw. Funktionalitäten sollten in gemeinsam genutzten Modulen umgesetzt werden. Das Kopieren von Funktionalität muss vermieden werden.

SOLID PRINZIPIEN – sind Prinzipien, die zu gutem objektorientierten Design führen sollen. Sie wurden neben anderen von Robert C. Martin, Bertrand Meyer und Barbara Liskov publiziert und propagiert.

SINGLE-RESPONSIBILITY-PRINZIP Es sollte nie mehr als einen Grund dafür geben, eine Klasse zu ändern.

OPEN-CLOSED-PRINZIP Module sollten sowohl offen (für Erweiterungen), als auch geschlossen (für Modifikationen) sein.

LISKOVSCHES SUBSTITUTIONSPRINZIP Das Liskovsche Substitutionsprinzip (LSP) oder Ersetzbarkeitsprinzip fordert, dass eine Instanz einer abgeleiteten Klasse sich so verhalten muss, dass jemand, der meint, ein Objekt der Basisklasse vor sich zu haben, nicht durch unerwartetes Verhalten überrascht wird, wenn es sich dabei tatsächlich um ein Objekt eines Subtyps handelt.

INTERFACE-SEGREGATION-PRINZIP Clients sollten nicht dazu gezwungen werden, von Interfaces abzuhängen, die sie nicht verwenden.

DEPENDENCY-INVERSION-PRINZIP Module hoher Ebenen sollten nicht von Modulen niedriger Ebenen abhängen. Beide sollten von Abstraktionen abhängen. Abstraktionen sollten nicht von Details abhängen. Details sollten von Abstraktionen abhängen.

(aus Wikipedia)

MODULAR DESIGN – Module stellen Baublöcke dar, die komplexere Funktionalitäten abstrahieren und anderen Teilen der Software durch Programmierschnittstellen (engl. APIs) verfügbar machen. Existierende Softwaremodule, welche die gewünschten Funktionalitäten bereits implementieren, sind ausreichend zu berücksichtigen.

5.2 Systembild / Software-Engineering

Komplexe Systeme und deren Komponenten sind durch gängige und geeignete Diagramme visuell darzustellen.

Für Prozesse sind z.B. Zustandsdiagramme oder BPM Diagramme zu erstellen und der Projektdokumentation beizufügen. Für Datenstrukturen sind entsprechend ER Diagramme anzufertigen. Bei ER Diagrammen ist auf vollständige Abbildung der Relationen und Kardinalitäten zu achten.

Die Diagramme sind in regelmäßigen Abständen zu aktualisieren.

5.3 Programmiersprache

Die Wahl der passenden Programmiersprache sollte hinsichtlich folgender Punkte erfolgen:

- Verfügbarkeit von Entwickelnden im Unternehmen als auch die Verfügbarkeit von Dienstleistenden Agenturen – Es dürfen keine Sprachen verwendet werden, für die Entwickelnde nicht verfügbar sind.
- Portabilität – Die Programmiersprache muss auf gängigen Systemen lauffähig sein. Insbesondere bei Webanwendungen sind portierbare Programmiersprachen zu wählen und ggf. Einschränkungen durch das Hosting im Vorfeld abzuklären.
- Verwendung starker Typisierung – Die Programmiersprache sollte die Datentypen mit abbilden, um vorhersehbare Fehler noch vor der Ausführung zu erkennen. Durch abbilden der erwarteten Datentypen wird der Quelltext zudem in Integrierten Entwicklungsumgebungen (IDEs) leichter Navigierbar.
- Verfügbarkeit – Es sind Programmiersprachen vorzuziehen, die frei Verfügbar sind.
- Effizienz – Bei der Wahl der Programmiersprache sollte eine Ressourcenschonende Datenverarbeitung ausreichend berücksichtigt werden.

5.4 Frameworks

Der Einsatz von Frameworks ist immer der individuellen Programmierung vorzuziehen. Bei der Evaluierung ist darauf zu achten, dass für das Framework ein langfristiger Support (LTS) angeboten wird. Wenn möglich ist die Laufzeit des Projekts zu berücksichtigen. Die Lizenz muss den kommerziellen Einsatz des Frameworks erlauben. Zusätzlich ist bei der Wahl zu beachten, dass Sicherheitsfeatures ein fester Bestandteil des Frameworks sind und damit die gängigen Angriffe abgewehrt werden können.

Es sollte bei der Wahl des Frameworks auch berücksichtig werden, dass die verwendeten Konventionen (z.B. MVC) sauber im Framework umgesetzt wurden und dass eine ausreichende Dokumentation vorliegt.

Es dürfen keine Veränderung an den Kerndateien des Frameworks durchgeführt werden, da dies ein schnelles Einspielen von Aktualisierungen unterbindet. Dies gilt auch für Bibliotheken von Dritten.

Bei der Wahl von Frameworks sollte die Ressourcenschonende Verarbeitung von Daten ausreichend Berücksichtigung finden.

Beispiel: Rendering, also das Erstellen des HTMLs für den Browser aus Daten auf dem Server, wird in modernen Single-Page-Webanwendungen in der Regel im Browser, und nicht wie bei klassischen Webanwendungen auf dem Server, realisiert. Dadurch reduziert sich zum einen die Komplexität der Serveranwendungen und ggf. die auf dem Server vorzuhaltenden Ressourcen.

Insbesondere bei Kryptografischen Funktionen oder Funktionen, die Benutzer-Zugangsdaten verarbeiten, sind entsprechende Standards und Frameworks (z.B. OAuth2, SAML, OpenID) zu verwenden.

5.5 Paketmanager

Software aus Paketmanagern ist stets selbst kompilierter Software vorzuziehen. Bei der Wahl der Paketmanager sollten folgende Punkte Berücksichtigung finden:

- Aktualität und Reaktionszeit hinsichtlich Sicherheitsupdates der verteilten Softwarepakete
- Abgleich der verwendeten Softwareversionen mit Vulnerabilitätsdatenbanken
- Erstellung von Software-Materiallisten (SBOM)
- Portabilität

5.6 Container

Bei der Verwendung von Containersystemen wie z.B. Docker ist zu berücksichtigen, dass das Betriebssystem des Containers zu einem Teil der Anwendung wird. Es sind entsprechende Maßnahmen zu ergreifen welche die Aktualität und Sicherheit des Betriebssystems und der Softwarekomponenten im Container sicherstellen.

Es sollte zudem darauf geachtet werden, dass Base Images verwendet werden, die regelmäßig Aktualisiert und deren Sicherheitslücken zeitnah geschlossen werden.

5.7 Versionierung

Es muss eine Versionierung des Programmcodes sowie der Features der Anwendung vorgenommen werden. Für die Versionierung des Programmcodes muss GIT als Software für die Versionskontrolle eingesetzt werden. Andere Versionskontrollsysteme bedürfen einer Genehmigung des Eigentümers dieser Richtlinie.

Die Trennung zwischen Test- und Produktivsystem [siehe Entwicklungs- und Testumgebung (Umgebungstrennung)] ist auf Ebene der Versionskontrolle (z.B. durch unterschiedliche GIT-Zweige) abzubilden. Es sind Maßnahmen zu ergreifen die verhindern, dass ungetestete bzw. nicht abgenommene Funktionen produktiv gehen.

Versionsnummern werden nach dem Semantic Versioning Standard (SemVer 2.0.0) gebildet.

https://semver.org/lang/de/#semantic-versioning-200

5.8 Tests

Für jedes neue Feature und jede Änderung an einem Feature der Anwendung müssen vor dem Einspielen Tests durchgeführt werden. Sollte es sich um eine kritische Anwendung handeln, müssen detailliertere Tests erfolgen.

Es wird zwischen Funktionstests, Verhaltenstests und Penetrationstests unterscheiden. Die ersten Tests können vom Entwickler durchgeführt werden, müssen aber vor der Freigabe oder des Ausrollens auf der

Produktivumgebung von einem zweiten Entwickelnden getestet werden. Die Verhaltenstests werden vom Auftraggeber vorgenommen. Möglichst von der Person oder dem Fachbereich, die die Anforderung eingereicht hat. Diese Person sollte die Anwendung regelmäßig nutzen.

5.8.1 Funktionstests (Unit-Tests)

Anwendungen sollten so geschrieben sein, dass man die Funktionalität auch ohne die Benutzerschnittstelle testen kann. Die Logik ist daher möglichst so zu kapseln, dass man in einem Unterprogramm ohne die grafische Benutzerschnittstelle verschiedene Szenarien (Testfälle) damit durchspielen kann.

In den Testfällen werden die einzelnen Komponenten so weit wie möglich separat voneinander getestet. Dafür sind, sofern verfügbar, entsprechende Testframeworks zu verwenden.

Funktionstests sollten am besten während oder direkt nach der Entwicklung der Funktion geschrieben werden. Zu Orientierung könnten die folgenden drei Schritte verwendet werden.

1. Schreiben Sie zuerst Tests für das erwünschte fehlerfreie Verhalten, für schon bekannte Fehlschläge bzw. Bugs oder für das nächste Teilstück an Funktionalität, das neu implementiert werden soll. Diese Tests werden vom bestehenden Programmcode erst einmal nicht erfüllt bzw. es gibt diesen noch gar nicht.
2. Ändern/schreiben Sie den Programmcode mit möglichst wenig Aufwand, bis nach dem anschließend angestoßenen Testdurchlauf alle Tests bestanden werden.
3. Räumen Sie dann im Code auf (Refactoring): Entfernen Sie Wiederholungen (Code-Duplizierung), abstrahieren Sie wo nötig, wenden Sie Code-Konventionen an etc. Natürlich wieder mit abschließendem Testen. Ziel des Aufräumens ist es, den Code schlicht und verständlich zu machen.

5.8.2 Verhaltenstests

Nach den Funktionstest muss getestet werden, ob das neue Feature den Anforderungen entspricht. Dies sollte von einem Mitarbeiter aus den Fachabteilungen oder vom Projektleiter durchgeführt werden. Dabei ist auch die Benutzerfreundlichkeit zu beachten.

Nach erfolgreichem Verhaltenstest gilt die Funktion / das Feature als abgenommen und der Testende bestätigt die Freigabe durch eine Zustandsänderung am entsprechenden JIRA-Ticket.

5.8.3 Penetrationstests

Bei kritischen Anwendungen, die vertrauliche Informationen verarbeiten, sollten in regelmäßigen Abständen Penetrationstests durchgeführt werden. Dies muss von Personen durchgeführt werden, die entsprechende Kompetenz nachweisen können.

5.9 Reviews

Es ist sicherzustellen, dass ausschließlich Code in das Produktivsystem überführt (Released) wird, der durch eine zweite Person gelesen und unter Zuhilfenahme der Dokumentation verstanden wurde.

Es müssen in regelmäßigen Abständen Code-Reviews durchgeführt und dabei gefundene Fehler beseitigt werden. Der Code sollte nach den festgelegten „Code Guidelines“ strukturiert sein.

5.10 Validierung

Alle Benutzereingaben müssen vor der ersten Verarbeitung validiert werden. Dabei spielt es keine Rolle, wer diese Eingaben tätigt. Die Eingaben müssen durch Validierung hinsichtlich folgender Angriffe abgesichert werden:

- Injections
- Cross-Site Scripting (XSS)
- Cross-Site Request Forgery (CSRF)
- Manipuliertes Formular
- Nullbytes (%00)
- Zeilenumbrüche (%0d, %0a, \n, \r)
- Pfadangaben (z.B. das Traversieren relativer Pfade wie ../ oder ..\)

Die Informationen im Header, in der URL und im Cookie müssen ebenfalls validiert werden.

Des Weiteren muss die Eingabe des Benutzenden auf die zu erwartende Eingabe überprüft werden. Wenn beispielsweise eine E-Mail-Adresse in das Eingabefeld eingetragen werden soll, muss die Eingabe auch daraufhin validiert werden.

Die Validierung muss immer serverseitig erfolgen und sollte im Programmcode an einer zentralen Stelle implementiert werden. Außerdem muss für jede Eingabe der gleiche Zeichensatz, wie z.B. UTF-8, festgelegt werden.

5.11 Aktualisierung

Aktualisierungen von Frameworks, Abhängigkeiten wie Modulen bzw. Bibliotheken und der verwendeten Dienste müssen zeitnah eingespielt werden.

5.12 Berechtigungskonzept

Es ist ein Berechtigungskonzept für jede Anwendung zu erstellen. Damit soll sichergestellt werden, dass jeder Benutzende nur die Zugriffe erhält, die er zur Erfüllung seiner Aufgabe benötigt (Need to know).

Ein geeignetes Rollen- und Berechtigungskonzept kann ggf. bereits aus den ermittelten Stakeholdern in der Entwurfsphase abgeleitet werden.

5.13 Authentifizierung und Passwort Management

Sensible Daten oder systemkritische Funktionen müssen immer durch eine Zugangskontrolle geschützt sein. Dies bedarf einer Authentifizierung der Benutzenden.

Es muss so verfahren werden, dass alle Seiten und Funktionen standardmäßig einer Authentifizierung bedürfen und dass öffentliche Seiten und Funktion explizit freigegeben werden müssen.

Wenn möglich, sollten geprüfte und verbreitete Standards für Authentifizierungsdienste genutzt werden (z.B. OAuth2, SAML, OpenID). Wenn der Authentifizierungsmechanismus selbst implementiert wird, muss dies an einer zentralen Stelle im Programmcode geschehen.

Die Authentifizierungslogik muss von den aufrufbaren Ressourcen getrennt und beispielsweise durch eine Umleitung geschützt werden. Falls die Authentifizierung scheitert, muss eine definierte Fehlermeldung zurückgegeben werden. Diese Fehlermeldung darf keinen Aufschluss darüber geben, welcher Teil der Authentifizierung fehlgeschlagen ist. In Webanwendungen dürfen nur HTTP-POST Anfragen zum Übertragen der Authentifizierungsdaten genutzt werden. Das Ver-senden von Passwörtern durch die Anwendung ist nicht erlaubt. Eine „Passwort Vergessen"-Funktion könnte über ein zufällig generiertes Token realisiert werden.

Die Anmeldemaske muss gegen einen Brute-Force Angriff abgesichert sein. Vorzugsweise sollte das Konto nach einer gewissen Anzahl von gescheiterten Anmeldeversuchen (z.B. nach fünf Versuchen) gesperrt und nach einer definierten Zeit wieder entsperrt werden. Die Zeit sollte sich nach jedem gescheiterten Versuch erhöhen. Weiterhin kann ein Captcha eingesetzt werden, wobei auf eine barrierearme Umsetzung zu achten ist.

Sollten für die Authentifizierung Komponenten von Drittanbietern genutzt werden, muss der Programmcode sorgfältig geprüft werden.

Passwörter dürfen nur als Hashwert gespeichert werden und sollten außerdem zusätzlich mit einem Salt versehen werden. Es dürfen keine veraltet Hash-Algorithmen wie z.B. der MD5-Algorithmus verwendet werden. Es wird der Hash-Algorithmus SHA256 empfohlen. Sollte es sich bei der Anwendung um ein Arbeitsmittel handeln, müssen die Anforderungen der Passwortrichtlinie technisch umgesetzt werden.

Zugangsdaten dürfen nur verschlüsselt übertragen werden. Passwörter sollten das System, für das sie vom Benutzenden angelegt wurden, nicht verlassen - d.h. Zugangsdaten sollten vom Systemdesign her nicht an Drittsysteme übermittelt werden müssen. In der Regel kann z.B. durch Zugriffscodes (Tokens) oder Challenges ein solches Verhalten per Design ausgeschlossen werden.

Die Passworteingabe des Benutzenden darf auf dem Bildschirm nicht zu sehen sein oder sollte durch Punkte oder Sterne ersetzt werden. Passwörter dürfen nicht in Logdateien erscheinen.

Falls temporäre Passwörter genutzt werden, müssen diese eine kurze Gültigkeit haben und es muss technisch sichergestellt werden, dass das temporäre Passwort bei der nächsten Nutzung gewechselt wird.

Passwörter müssen entsprechend der Passwortrichtlinie in regelmäßigen Zeitabständen gewechselt werden. Dabei muss gewährleistet werden, dass nicht erneut das gleiche Passwort verwendet wird. Weiterhin darf das Passwort nur einmal pro Tag gewechselt werden, um Angriffe auf diese Funktion vorzubeugen. Ein Passwortwechsel muss dem Nutzer immer gemeldet werden, damit der Nutzer reagieren kann, falls der Passwortwechsel nicht vom ihm angestoßen wurde.

Das Datum der letzten Systemanmeldung durch den Nutzer muss gespeichert werden. Passwörter werden häufig durch den Nutzer im Browser gespeichert. Wenn technisch umsetzbar, ist dies zu verhindern.

Am Ende der Sitzung muss der Nutzer die Möglichkeit haben sich abzumelden. Die Abmeldefunktion muss überall in der Anwendung aufrufbar sein.

Inaktive Nutzer sollten nach einer gewissen Zeit abgemeldet werden.

5.14 Session Management

Bei webbasierten Anwendungen sind Sessions die bevorzugte Lösung, Daten über mehrere Aufrufe zu speichern. Das HTTP-Protokoll kann dies nicht leisten, da es ein zustandsloses Protokoll ist.

Bevor eine neue Session erstellt wird, ist darauf zu achten, dass eine bereits existierende Session beendet wird. Wenn der Nutzer sich von der Anwendung abmeldet, muss die Sitzung vollständig beendet und die Session gelöscht werden. Eine dauerhafte Anmeldung an einer Anwendung ist nicht erlaubt und muss durch eine automatische Abmeldung nach einer definierten Zeit verhindert werden. Dies gilt besonders für kritische Anwendungen, die sensible Informationen verarbeiten.

Session Cookies sollten immer das „secure“ und “HttpOnly” Attribut beinhalten. Das „HttpOnly“ Attribut darf nur bei Anwendungen weggelassen werden, bei denen das Cookie clientseitig ausgelesen werden muss.

Im Normalfall sollte das Session Management vom Server oder nachrangig vom Framework genutzt werden. Falls dies nicht möglich ist, müssen folgende Punkte beachten werden:

- Es muss ein gut geprüfter Algorithmus zur Erstellung der Session ID eingesetzt werden, um eine ausreichend zufällige ID zu generieren.
- Es muss der Pfad und die Domain entsprechend der Webseite im Session Cookie angegeben werden.
- Für jede neue Authentifizierung muss eine neue Session ID erzeugt werden.
- Session IDs dürfen nur im Http-Cookie-Header übertragen werden (Präventivmaßnahme gegen Session-Hijacking).
- Die Session Daten müssen vor unbefugten Zugriff geschützt werden.
- Die Session IDs sollten in kurzen, definierten Abständen neu erstellt und die alten IDs deaktiviert werden (Präventivmaßnahme gegen Session-Hijacking).

5.15 Zugriffskontrolle

Der Zugriff auf Ressourcen wie z.B. Funktionen, Dateien, URLs, Services sowie anderen sicherheitsrelevanten Diensten muss durch eine Kontrollinstanz abgesichert sein, d.h. jede Funktion und jeder Dienst muss explizit für eine Person oder eine Benutzergruppe freigegeben werden (Whitelist). Falls der Zugriff durch mangelnde Berechtigungen scheitert, muss dem Nutzer eine definierte Fehlermeldung angezeigt werden. Des Weiteren muss die Berechtigung des Nutzers bei jedem Aufruf überprüft werden.

Die Regelung des Zugriffes sollte an zentraler Stelle im Programmcode erfolgen bzw. übersichtlich gestaltet sein und muss serverseitig stattfinden. Die Zugriffskontrolle darf nicht über den „referer“ Header erfolgen.

Es sollten Geltungsbereiche (Scopes) für den Zugriff auf Ressourcen definiert werden.

Bei der Wahl der Zugriffskontrollmechanismen sollte ressourcenschonende und performante Datenverarbeitung berücksichtigt werden.

5.16 Fehlerbehandlung

Für jeden bekannten Fehler, wie z.B. „Zugriff verweigert“ oder „Seite nicht gefunden“, muss eine definierte Fehlermeldung zurückgegeben werden. Für jeden unbekannten Fehler muss eine Standardfehlermeldung generiert werden. Die Fehlermeldung sollten dem Nutzenden erklären, wie weiter zu verfahren ist. Es dürfen darüber hinaus keine weiteren Angaben über die Art oder den Ort des Fehlers gemacht werden. Die Devise ist, so wenige Informationen wie möglich herauszugeben.

5.17 Logging

Jeder Fehler, Angriff oder fehlgeschlagene Zugriff auf Ressourcen der Anwendung muss protokolliert werden. Der Zugriff auf die LOG-Dateien ist auf die Administrierenden und Entwickelnden zu beschränken.

5.17.1 Sicherheitsverstöße

Es sollten Maßnahmen zur Erkennung von Sicherheitsverstößen vorgenommen werden. Beispielsweise sollten fehlgeschlagene Anmeldeversuche mit geeigneten Werkzeugen wie Fail2Ban protokolliert und entsprechende Konten oder IPs zeitweise gesperrt werden.

5.17.2 Personenbezogene Daten

Personenbezogene Daten (z.B. IP-Adresse, Passwort) oder andere sensible Informationen dürfen nicht bzw. nur anonymisiert protokolliert werden. Hier gilt die aktuelle gesetzliche Lage.

5.18 Datenintegrität

Sensible Dateien Nutzender oder wichtige Systemdateien, wie z.B. Konfigurationsdateien sollten durch eine Prüfsumme, die separat z.B. in einer Datenbank gespeichert wird, vor Manipulationen geschützt werden. Falls eine Datei manipuliert wurde, sollten Nutzende vor dem Herunterladen gewarnt werden.

5.19 Datenbanken

Alle Eingaben, die von Nutzenden weiter zur Datenbank gereicht werden, wie z.B. Suchanfragen, müssen sorgfältig validiert werden. Auch hier gilt, dass die Verwaltung der Datenbankverbindung über das Framework abgewickelt werden sollte. Zudem sollte für jedes Programm und jede Anwendung ein eigenes Datenbankkonto mit angemessen eingeschränkten Rechten verwendet werden. Alle unnötigen Datenbankberechtigungen sowie -funktionen sollten abgeschaltet werden.

5.20 Passwörter

Alle Passwörter, die die Anwendung verwendet (z.B. für Datenbank oder Verschlüsselung) müssen entsprechend der Passwortrichtlinie (siehe Bezugsdokumente) gewählt werden.

5.21 Systemkonfiguration

Folgende Punkte müssen in der Systemkonfiguration beachtet werden:

- Ausschalten von „directory listings“ auf Webservern
- Beschränken der Rechte des Webservers, Prozesses bzw. Dienstes auf ein Minimum
- Fehler müssen immer kontrolliert zurückgegeben werden (siehe Fehlerbehandlung)
- Alle unnötigen Funktionen oder Dateien müssen entfernt werden
- Testprogrammcode oder Funktionalitäten, die für den Produktivbetrieb nicht benötigt werden, müssen entfernt werden
- Um die Verzeichnisstruktur nicht offen zu legen, sollten in der robots.txt keine einzelnen Ordner freigegeben oder blockiert werden. Hiervon ausgenommen ist das Elternverzeichnis. Dazu sollten sich alle freigegebenen oder blockierten Ordner in einem Verzeichnis befinden, welcher dann in der robots.txt freigeschaltet oder blockiert wird.
- Zugangsdaten dürfen nicht in die Versionskontrolle gelangen und dürfen daher niemals im Quellcode abgelegt werden. Ggf. sind Konfigurationsframeworks wie z.B. DotEnv zu verwenden. Wenn Zugangsdaten in die Versionskontrolle gelangen, sind diese als kompromittiert zu betrachten.

5.22 Dateiuploads

Dateien, die über die Anwendung hochgeladen oder verarbeitet werden, sollten nur an einem Ort gespeichert werden. Dateien mit sensiblem Inhalt dürfen nicht ohne Zugriffskontrolle erreichbar sein. Das gilt auch für das Hochladen einer Datei.

Weiterhin ist Folgendes zu beachten:

- Es dürfen nur zugelassene Dateitypen (Whitelist) auf den Server hochgeladen werden. Die Überprüfung läuft über den Dateiheader. Die Überprüfung der Dateiendung allein reicht nicht aus.
- Die Dateien der Nutzenden müssen von den Programmcodedateien getrennt sein.
- Das Hochladen von Dateien, die der Webserver möglicherweise interpretieren kann, muss unterbunden werden.
- Auf dem Upload-Ordner müssen die Ausführungsrechte abgeschaltet sein.
- Der absolute Pfad darf nicht an Nutzende gesendet werden.
- Hochgeladene Dateien müssen nach Viren und Malware durchsucht werden.

5.23 Speicher- bzw. Ressourcenverwaltung

Das Speichermanagement sollte immer vom Server oder Framework übernommen werden. Falls dies nicht der Fall ist, sind folgende Punkte zu beachten:

- Pufferspeicher müssen ausreichend groß sein.
- Programmschleifen dürfen den Speicher nicht zum Überlaufen bringen. Alle Schleifen sind mit Abbruchkriterien zu versehen.
- Alle Zeichenketten müssen auf eine festgelegte Länge gekürzt werden, bevor sie kopiert oder mit anderen Zeichenketten verbunden werden.
- Ressourcenobjekte, wie z.B. Verbindungsobjekte oder Datei-Handles, müssen spätestens am Ende des Programms geschlossen werden. Die Löschung solcher Objekte durch den Garbage Collector ist keine Alternative.

- Wenn verfügbar, müssen nicht ausführbare Stacks verwendet werden.
- Am Ende des Programms muss der reservierte Speicher wieder freigegeben werden.

5.24 Programmierkonventionen (Coding Guidelines)

In jedem Projekt sollten Code Guidelines festgelegt, oder auf bereits etablierte Guidelines zurückgegriffen werden. In den Guidelines sollten folgende Punkte festgelegt werden:

- Verwenden der üblichen Vorgehensweisen im gewählten Programmierparadigma (z. B. Objektorientierte Programmierung)
- Festlegung von Namenskonventionen: Wie sind Bezeichner zu wählen?
- Verwendung von Compilerdirektiven und -schaltern
- Strukturierung des Codes (Einrückungen, Modul-/Prozedurgröße, GOTO-Verbot): Wo sollen Leerzeichen stehen? Wie ist einzurücken? Maximale Zeilenanzahl einer Routine.
- Typisierung (Wahl des Typs für ein Symbol oder eine Variable)
- Initialisierung von Variablen
- Zugriff auf Variablen fremder Objekte / Prozeduren
- Gestaltung von Funktionsaufrufen (Parameterübergaben, Rückgabewerte)
- pflichtgemäß zu verwendende Standardkomponenten, wie Unterprogramme, APIs etc.
- Vermeidung von Redundanz und möglichst breite Wiederverwendbarkeit durch Modularisierung
- Unabhängigkeit verschiedener Programmteile (Modularität)
- Einheitlichkeit bei der Lösung gleichartiger Probleme, z. B. durch Normierte Programmierung
- Robustheit durch ausführliche Fehler- und Ausnahmebehandlung
- Umfang und Form der Dokumentation: Je Prozedur, je Zeile; Detaillierungsgrad; abgestimmt auf weitere Dokumente

5.25 Kryptographie und Kommunikation

Die Richtlinie zur Verwendung kryptografischer Verfahren, in der IT-Sicherheitsrichtlinie (siehe Bezugsdokumente) ist zu beachten.

6 Deployment

Für das automatische Ausrollen der Änderungen auf Test- und Produktivservern setzt der Bereich VDI-IT-Anwendungsentwicklung Jenkins ein. In Jenkins wird für jedes Projekt ein separater Ordner erstellt der die Deployment-Jobs sowie die Zugangsdaten für das Projekt enthält.

Jeder Server muss eigene Zugangsdaten verwenden. Passwörter werden gemäß der Passwortrichtlinie (siehe Bezugsdokumente) zufällig generiert. Wo möglich, muss SSH mit Schlüsselbasierter Anmeldung verwendet werden, um die Anzahl der zu öffnenden Ports so gering wie möglich zu halten und somit Angriffsvektoren zu minimieren.

Zugangsdaten, die nur während des Deployments verwendet werden, müssen auf dem Jenkins Server verwaltet und vor Zugriff durch andere Projekte gesichert werden.

7 Dokumentation

Für jeden Punkt unter „Entwicklung“ muss das Vorgehen dokumentiert werden. Des Weiteren muss eine technische Dokumentation erstellt werden. Wenn möglich sind dabei Dokumentationswerkzeuge wie z.B. Doxygen zu verwenden. Dazu ist es nötig, dass alle öffentlichen Funktionen kommentiert werden. Die Kommentare müssen in kurzen Sätzen den Zweck der Funktion erläutern. Darüber hinaus müssen Übergabe- und Rückgabeparameter definiert werden. Dazu sollten, wenn möglich, Annotationen genutzt werden.

7.1 Benutzerhandbuch

Für die Liefergegenstände ist ein Benutzerhandbuch zu erstellen.

7.2 Technische Dokumentation

In der Technischen Dokumentation ist ggf. auf den Projektsteckbrief zu verweisen. Bei Verweisen auf externe Dokumentationen Müssen diese dem Auftraggeber zur Verfügung gestellt werden. Auch hier gilt, dass die Dokumentation dieser Richtlinie entsprechen.

7.3 Server Dokumentation

Die verwendeten Webserver werden in der Webserverdokumentation auf dem VDI-SharePoint dokumentiert.

8 Sicherheit der Entwicklungsumgebung

Für die Softwareentwicklung wird eine Entwicklungsumgebung benötigt. Es darf nicht im Produktivsystem entwickelt werden. Das Vorgehen in den folgenden Punkten muss dokumentiert werden.

8.1 Backup

Es muss sichergestellt werden, dass die Daten in der Entwicklungsumgebung regelmäßig gesichert werden.

8.2 Zugriff

Der Zugriff auf den Programmcode, auf die Datenbank und auf das Repository muss auf die Entwickler beschränkt werden.

8.3 Testdaten

Zum Entwickeln dürfen nur Testdaten und keine Echtdaten verwendet werden. In den Testdaten dürfen keine sensiblen Daten enthalten sein. Diese müssen im Vorfeld anonymisiert werden.

8.4 Funktionstrennung

Bei jedem Projekt ist zu entscheiden, ob eine klare Funktionstrennung vollzogen werden kann. Klare Funktionstrennung bedeutet, dass ein Programmierer nicht das Recht und die Aufgabe haben darf, den von ihm erstellten Programmcode auf die Liveumgebung zu spielen. Dies muss nach einer Prüfung von einer Person umgesetzt werden, die an der Erstellung des Programmcodes nicht beteiligt war. Dies vermeidet nach dem Vier-Augen-Prinzip Fehler und erschwert ungewollte Manipulationen.

8.5 Entwicklungs- und Testumgebung (Umgebungstrennung)

Es darf nicht im Produktivsystem entwickelt werden. Eine Umgebungstrennung zwischen Produktiv- und Test- bzw. Staging-System ist, sofern anwendbar, über den gängigen Standard „GIT-Flow“ auf Ebene des Quellcodes in der Versionskontrolle abzubilden.

Die Änderungen werden in Hotfix- oder Featurezweigen geführt, so dass Sie einzeln erst in das Test- und danach in das Produktivsystem übernommen werden können.

Beim Einchecken von Features bzw. Hotfixes in die Versionskontrolle ist in einer Commit-Nachricht Bezug auf den auslösenden Vorgang im Ticketsystem zu nehmen und die Änderung kurz in englischer Sprache zu beschreiben.

Bei Systemen, bei denen kein Versionskontrollsystem verwendet werden kann, sind die Änderungen durch Kommentare im Quellcode kenntlich zu machen. Es müssen das Datum der Änderung, Kürzel der Ändernden Mitarbeitenden sowie die Ticketnummer des auslösenden Vorgangs enthalten sein.

Beim Deployment wird der Umgebungstrennung durch die folgenden Verfahren Rechnung getragen.

- Für Anwendungen, die auf den Rechnern von Mitarbeitenden ausgeführt werden, sind vor dem Ausrollen der Produktivversion im Rahmen eines Beta-Tests die neuen Funktionalitäten in einer Vorabversion von vorher ausgewählten Testenden zu prüfen.
- Verteilte Systeme bzw. Webanwendungen müssen ein vom Produktivsystem getrenntes, gespiegeltes Staging- bzw. Testsystem für Tests und Abnahme zur Verfügung gestellt bekommen. Erst wenn ein Feature dort getestet und freigegeben wurde, darf es in das Produktivsystem übernommen werden.

Hotfixes bekommen Vorrang und müssen in agilen Projekten nicht auf Abschluss des Sprints warten, sondern dürfen einzeln in das Produktivsystem übernommen werden. Sonst ist der Ablauf der gleiche – auch Hotfixes werden getestet.

Eine versehentliche Übernahme von ungetesteten Features ist nach Möglichkeit durch technische Vorkehrungen zu verhindern (z.B. [a] Änderungen am Master-Zweig nur durch Pull-Requests zulassen oder [b] Gültigkeitsprüfungen in Commit-Hooks z.B. mit CaptainHook).

Nach Möglichkeit ist die Übernahme in die Produktivumgebung mit einem Review verbunden und durch eine andere Person als dem Entwickelnden durchzuführen. Dabei ist auch die Vollständigkeit der Dokumentation zu prüfen.

Im Testsystem darf nicht mit Echtdaten, sondern nur mit Testdaten gearbeitet werden. Bei Bedarf sind Testdaten durch geeignete Werkzeuge (z.B. Faker Bibliothek für diverse Programmiersprachen) zu generieren. Dennoch sollten die Testdaten in Form, Qualität und Limits die gleichen Eigenschaften wie die Echtdaten besitzen.

9 Entwickelnde

Vor der Arbeit an einem Softwareentwicklungsprojekt muss gewährleitet sein, dass die Software-Entwickelnden die entsprechende Kompetenz und Ausbildung aufweisen. Darüber hinaus sollten die Software-Entwickelnden die Möglichkeit haben sich zu dem Thema weiterzubilden.

Eine Trennung zwischen Administratoren mit Verwaltungsrechten und Software-Entwickelnden mit Bearbeitungsrechten ist ggf. in Betracht zu ziehen. Es sollte gewährleistet werden können, dass Konten für Entwickelnde nicht über Verwaltungsrechte verfügen, mit denen in Drittsystemen Schaden angerichtet werden könnte.

10 Bezugsdokumente

IT-Sicherheitsrichtlinie - IT-Sicherheitsrichtlinie_VDI_GmbH_V1.00.pdf

Passwortrichtlinie - Passwortrichtlinie_VDI_GmbH_V1.00.pdf