ISMS – Informationssicherheitsmanagement

Richtlinie zur Softwareentwicklung

Vertraulichkeit: EuroDaT-intern[1]

EuroDaT GmbH
Gerichtsstraße 2
65185 Wiesbaden
Germany

info@eurodat.org
www.eurodat.org

[1] Weitergabe an Kunden auf Anfrage möglich.

Dokumenteninformation	
Version:	0.1
Verantwortlich:	Alexander Alldridge
Freigabe:	Freigabe erfolgt am 10.12.2024 durch die Alexander Alldridge
Nächste Überprüfung:	10.12.2025

Änderungshistorie:

Version	Datum	Bearbeiter	Änderungshinweise
0.1	31.10.2024	Lukas Klose	Erstellung initiale Draft-Version
0.2	19.11.2024	Lukas Klose	Einarbeitung Kommentare Arthur Wigandt
0.3	10.12.2024	Alexander Alldridge	Kleinere Korrekturen; Freigabe erteilt

Inhaltsverzeichnis

1 Einleitung

Die Entwicklung von Software ist ein wichtiger Bestandteil der Leistung von EuroDaT. Dabei besteht der Großteil der Softwareentwicklung aus Entwicklungsteams, die an der Software für die EuroDaT-Plattform und teils auch für Applikationen für die Verwendung mit der EuroDaT-Plattform arbeiten. Diese Richtlinie setzt Vorgaben für die Erstellung, den Test und die Einführung von Software bei EuroDaT.

Diese Richtlinie gilt nicht für externe Anwendungsentwickler, die nicht im Auftrag von EuroDaT Applikationen für die EuroDaT-Plattform entwickeln. In diesem Fall liegt die Verantwortung zur Einhaltung von Sicherheitsrichtlinien und Qualitätsstandards nicht bei EuroDaT.

2 Gegenstand des Dokuments

Diese Richtlinie zur Softwareentwicklung betrifft jede Softwareentwicklung, die von EuroDaT beauftragt wird. Dies kann Entwicklungsarbeiten an der EuroDaT-Plattform, aber auch Entwicklungsarbeiten an Applikationen für die Nutzung innerhalb der EuroDaT-Plattform betreffen.

Externe Parteien, die Applikationen für die EuroDaT-Plattform ohne Beauftragung durch EuroDaT entwickeln, sind von dieser Richtlinie nicht betroffen.

2.1 Eingrenzung für interne Projekte

Um der Praxis Gerecht zu werden, dass Entwicklungsaktivitäten im Rahmen interner Test- oder Proof-of-Concept-Ansätze gestartet werden, können solche initialen Ansätze umgesetzt werden, ohne zwangsläufig alle Sicherheitsanforderungen umzusetzen. Voraussetzung für die Entwicklung ohne das Einhalten aller Sicherheitsanforderungen ist, dass der Code nicht ohne Einhaltung der Maßnahmen in Produktivsystemen eingesetzt wird.

Dabei ist darauf zu achten, dass keine schützenswerten Daten in Software verwendet wird, die ohne Einhaltung aller Sicherheitsanforderungen umgesetzt wurde. Wenn das Nutzen von schützenswerten Daten in einem internen Projekt nicht vermieden werden kann, gilt es, die Risiken zu identifizieren und technische sowie organisatorische Maßnahmen zur Risikoreduzierung umzusetzen.

3 Vorgaben für sicheres Coding

Codingstandards sind für jedes Projekt festzuhalten und automatisiert zu überprüfen (zum Beispiel in Gitlab-Pipelines). Dabei ist sich an üblichen Standards der jeweils verwendeten Programmiersprachen zu halten. Zusätzlich sollten Vorgaben für Namensschemata für Branches, Commit-Messages usw. festgehalten werden. Die Richtlinie gibt keine speziellen Anforderungen an die für jedes Softwareprojekt zu verwendeten Coding-Standards. Es ist jedoch sinnvoll, die Standards zwischen den Projekten konsistent zu halten, um einen Transfer zwischen verschiedenen Projekten zu ermöglichen.

3.1 Einhalten von Qualitätsstandards und Ausbau der Code-Sicherheit

Zum Einhalten von Qualitätsstandards bereits bei der Erstellung von Code, sollte eine IDE zum Programmieren benutzt werden, die bereits Coding-Standards bzw. die Entwicklung eines „sauberen Codes“ überprüft. Einstellungen von Linter in der Entwicklungsumgebung sollten so vorgenommen werden, dass sie mit der automatisierten weiteren Überprüfung in Gitlab-Pipelines in Einklang sind. Code sollte immer nach dem Vier-Augen-Prinzip aufgenommen werden. In der Umsetzung heißt die meist, dass die Aufnahme von Code (commit) von einer anderen Person erfolgt als die Übernahme von Code in weitere Branches (merge).

Die einzelnen Komponenten einer Software werden über kontinuierliche Integration (CI) zusammengefügt und der Code wird auch automatisierten Tests innerhalb des CI-Tools (Gitlab) unterworfen (siehe auch Kapitel 3.3 Softwaretests).

3.1.1 Vorgaben zum Steigern der Codesicherheit

Um die Bedrohungen durch Sicherheitslücken in entwickelten Anwendungen zu minimieren, werden mehrere Maßnahmen umgesetzt. Diese betreffen nicht nur sicheren Code, sondern auch das Deployment von Anwendungen:

- Mögliche Bedrohungen für ein zu entwickelndes System werden vorher anhand eines Bedrohungsmodells ermittelt. Dieses Modell wird über den Fortgang des Projektes weiterentwickelt und mit sich ändernden Anforderungen angepasst.[2]
- Umgebungen zum Testen, Entwickeln und für den Produktiveinsatz werden stets getrennt. Test- und Entwicklungsumgebungen sollten keine sensitiven Daten enthalten.
- Bei der Entwicklung wird mittels geeigneter Checklisten (z.B. OWASP ASVS[3]) geprüft, ob Best Practices eingehalten wurden. Dabei kann der Fokus auch auf ausgewählte Teile der Checkliste entsprechend des Bedrohungsmodells gelegt werden.
- Eingaben in ein von EuroDaT entwickeltes System werden validiert. Hierbei kann ein White-Listing-Verfahren eingesetzt werden, dass nur vorgegebene Eingaben akzeptiert.
- Innerhalb der CI/CD-Pipeline werden automatische Analyseverfahren zum Testen der

[2] Siehe auch OWASPs Guidelines zu DevSecOps

[3] https://owasp.org/www-project-application-security-verification-standard/

Anwendung eingerichtet (siehe SAST, DAST oder IAST). Dies bietet eine kontinuierliche Überprüfung des Quellcodes auf Schwachstellen und erlaubt es, Schwachstellen möglichst früh im Entwicklungsprozess zu erkennen. Hierzu wird mindestens eine statische Codeanalyse durchgeführt werden, wobei bei hohem Schutzbedarf diese durch dynamische oder interaktive Testverfahren erweitert werden kann.[4]

3.1.2 Verwalten von Abhängigkeiten

Eingebundene Bibliotheken und Softwarepakete müssen, wenn möglich, aus vertrauenswürdigen Quellen bezogen werden und in jedem Fall auf Schwachstellen geprüft werden. Dabei wird beim Hinzufügen einer neuen Abhängigkeit immer geprüft, ob diese für das Softwareprojekt notwendig ist. Für alle Abhängigkeiten muss für die Lizenzbedingen verifiziert werden, dass diese nicht durch den Einsatz von EuroDaT verletzt werden.

Ein Schwachstellenscan sollte automatisiert erfolgen. Das Scannen nach Schwachstellen sollte innerhalb von CI/CD-Pipelines eingeplant werden. Dies kann eventbezogen (zum Beispiel nach einem Commit/Merge) oder in regelmäßigen Abständen erfolgen (zum Bespiel automatisch 1x pro Tag).

Für das Scannen nach Schwachstellen bieten sich Lösungen an wie das Static Application Security Testing (SAST)[5] von Gitlab. Für das automatische Aktualisieren von abhängigen Paketen können Tools wie Renovate[6] verwendet werden. Dabei müssen alle Updates vor dem Einspielen in Prod getestet werden.

3.1.3 Sicherheitsmaßnahmen für Webseiten und -anwendungen

Webanwendungen verwenden heutzutage häufig Inhalte, die direkt von Webservern Dritter bezogen werden (bspw. Inhalte von CDNs). Dies wirft ähnliche Bedrohungen wie Bibliotheken und Pakete auf, welche eventuell schadhaften JavaScript Code in die Anwendung einschleusen. Um dies zu verhindern, sollen mehrere Sicherheitsmaßnahmen in Kombination genutzt werden, die das Risiko für einen Supply-Chain-Angriff effektiv verringern können. Folgende Maßnahmen können hierzu eingesetzt werden:

- Einschränken von Quellen für JavaScript Code. Die Content-Security-Policy (CSP) erlaubt es mittels der „script-src“-Direktive das Inkludieren von Skripten ausschließlich auf vorgegebene Quellen zu beschränken.
- Sicherstellen der Integrität von geladenen Ressourcen. Subresource Integrity (SRI) ermöglicht es die Integrität einer Ressource mittels Hashing zu überprüfen.[7]
- Isolation von JavaScript Code. Skripte, die über einem „sandboxed“ iframe-Element geladen werden, werden in einem getrennten Kontext mit eingeschränkten Privilegien

[4] NIST pflegt eine aktuelle Liste von Source Code Security Analyzers

[5] https://docs.gitlab.com/ee/user/application_security/sast/

[6] https://docs.renovatebot.com/

[7] Siehe auch https://www.srihash.org/

ausgeführt.

Jede öffentlich erreichbare Webschnittstelle, die von EuroDaT betreut oder entwickelt wird, sollte Informationen bereitstellen, die das Melden von Sicherheitslücken vereinfacht. Hierzu sollte der vom BSI empfohlene Standard RFC9116 verwendet werden. Zur Umsetzung wird eine Datei unter dem Namen security.txt unter dem Wurzelverzeichnis ({protocol}://{domain}/security.txt) oder in einem Verzeichnis für bekannte Dateien ({protocol}://{domain}/.well-known/security.txt) abgelegt.

Der Inhalt sollte wie folgt aufgebaut sein:

```
Contact: mailto:ciso@eurodat.org
Expires: 2028-12-31T00:00:00z
```

Die Gültigkeit sollte auf vier Jahre begrenzt werden, insofern keine anderen Vorgaben existieren. Projektteams können hierbei auf die Instanz der EuroDaT-Website verlinken: https://eurodat.org/security.txt.

3.1.4 Einsatz kryptographischer Verfahren

Beim Einsatz kryptographischer Verfahren wird geprüft, dass die jeweiligen Verfahren als sicher gelten und mit entsprechend empfohlenen Parametern, wie beispielsweise ausreichend lange Schlüssel, verwendet werden (siehe hierzu die Empfehlungen des BSI[8]). Dabei muss geklärt werden, wo Verschlüsselung eingesetzt wird und wie entsprechende Rahmenbedingungen geschaffen werden (z.B. Anschaffung von Zertifikaten, Schlüsselmanagement, sicheres Löschen von Schlüsseln, usw.[9]). Es sollte zudem regelmäßig geprüft werden, ob verwendete Verfahren und Parameter noch als sicher gelten.

Passwörter, Schlüssel, API-Keys oder ähnliche sicherheitsrelevante Zugangsdaten dürfen nicht im Quellcode gespeichert werden. In Gitlab können Umgebungsvariablen für das Speichern von Zugangsdaten verwendet werden. Diese sollten, wenn möglich, nicht im Klartext in Logs auftauchen. Gitlab maskiert die Variablen automatisch, wenn diese als „Masked“ markiert wird. Eine weitere Möglichkeit, Geheimnisse zu speichern, ist der Secret Manager in der Google Cloud Plattform. Dieser sollte bevorzugt werden, wenn die Daten nicht in der CI/CD-Pipeline benötigt werden.

3.1.5 Protokollierung und Monitoring

Fehlendes oder ungenügendes Protokollieren und Monitoring ist ein häufiges Problem, was dazu führen kann, dass Softwarefehler oder Angriffe zu spät entdeckt werden. Für Softwareentwicklungen gilt es deshalb, anhand des Schutzbedarfes zu betrachten, was protokolliert werden muss, welche Schritte zur Überwachung sinnvoll sind und dass

---

8 https://www.bsi.bund.de/EN/Themen/Unternehmen-und-Organisationen/Standards-und-Zertifizierung/Technische-Richtlinien/TR-nach-Thema-sortiert/tr02102/tr02102_node.html

9 Siehe hierzu Con.1 A1 bis A18 im BSI-Kompendium

ausreichende organisatorische und technische Voraussetzungen geschafften werden.

Für Softwareprodukte bei EuroDaT muss geklärt werden, inwiefern Protokollierung und Monitoring erforderlich ist und welche Daten erhoben werden sollen. Dabei ist zu beachten, dass Aktivitäten nicht nur zum Zweck der Fehlersuche und Systemauslastung während der Entwicklung protokolliert werden, sondern dass auch sicherheitsrelevante Ereignisse in Betracht gezogen werden. Das Cheatsheet von OWASP[10] und der Mindeststandard des BSI zur Protokollierung und Detektion von Cyberangriffen[11] geben einige solcher Ereignisse vor, die auf den entsprechenden Kontext angewandt werden können. Es sollte festgelegt werden, welche sicherheitsrelevante Benachrichtigungen eingerichtet werden sollen und wer für die Auswertung zuständig ist.

3.1.5.1 Schutz von Protokollierungsdaten und -systemen

Protokolldaten sollten gegen unberechtigte Veränderungen und Datenverlust geschützt werden. Dies kann u.a. mittels der folgenden Maßnahmen erreicht werden:

- IT-Systeme, die Protokolldaten erzeugen, verwenden synchronisierte Systemzeiten.
- Die Integrität von gespeicherten Protokolldaten wird mittels geeigneter Verfahren geschützt (bspw. mittels „append-only"-Zugriff, Übertragung auf „read-only“-Speichermedien, Hashing und Privilegien-Beschränkung).
- Protokolldaten werden codiert, um Injection Attacks zu mittigeren.
- Protokolldaten werden nach einer festgelegten Dauer archiviert.
- Wenn Protokolldaten übertragt werden, wird die Vertraulichkeit und Integrität auf dem Übertragungsweg gesichert.

Um die Voraussetzungen an die Protokollierungsdaten zu erfüllen, wird von der EuroDaT-Plattform unter anderem das in der Google Cloud Console integrierte Logging verwendet. Die Konfiguration hat dabei so zu erfolgen, dass die genannten Anforderungen an die Protokollierung beim Speichern in Cloud Storage Buckets und beim Abrufen im Logs Explorer erfüllt werden.

3.1.5.2 Einhaltung von Datenschutz

Protokolldaten können eine datenschutzrechtliche Relevanz aufweisen, welche es zu beachten gilt. Im Kontext eines Webservers ist es beispielsweise sinnvoll die IP-Adresse, die Anfrage und Zeit einer Anfrage zu speichern. Da es sich bei IP-Adressen um personenbezogene Daten handelt, muss eine Protokollierung immer zweckgebunden stattfinden (bspw. Sicherheit oder Bereitstellung des Dienstes) und darf nicht für andere Zwecke verwendet werden (z.B. Analytics).

Die Erzeugung von Protokollierungsdaten wird mit der ISB bzw. der Geschäftsführung abgestimmt. Hierbei muss zwingend festgehalten werden, welche Daten in welchem Umfang erhoben werden und nach welcher Frist sie wieder zu löschen sind. Existieren keine

[10] https://cheatsheetseries.owasp.org/cheatsheets/Logging_Cheat_Sheet.html#which-events-to-log

[11] https://www.bsi.bund.de/DE/Themen/Oeffentliche-Verwaltung/Mindeststandards/PDCA/PDCA.html

gesetzlichen Vorgaben, die die Speicherfrist von Protokollierungsdaten begrenzen, sollten diese mit Ablauf der Zweckgebundenheit entfernt werden.

3.1.6 Repositories und Versionierung

Zur Versionsverwaltung des Source-Codes wird bei internen sowie externen Projekten GitLab verwendet. Der Open-Source-Code von EuroDat wird unter https://gitlab.com/eurodat/ verwaltet. Darüber hinaus werden auch Continuous Integration und Continuous Delivery (CI/CD) über GitLab durchgeführt.

Die verschiedenen Versionen einer Software werden stets kenntlich gemacht, um eine Nachvollziehbarkeit der Releases sicherstellen zu können. Jeder Release sollte als Major, Minor oder Patch eingestuft werden. Abweichungen hiervon lassen sich ggf. durch die Art der Software begründen. Dabei wird für jedes Softwareprojekt festgelegt, welche Aktivitäten in der jeweiligen Releasestufe auszuführen sind.

3.2 Containerisierung

Häufig wird in EuroDaTs Softwareprojekten eine Containerisierung der Software verwendet. Containerisierung ist dabei das Verpacken von Softwarecode in Pakete (Paketierung), die die sämtlichen erforderlichen Komponenten wie Libraries, Frameworks und andere Abhängigkeiten enthalten und in ihrem eigenen „Container" isoliert sind. Containerisierte Anwendungen bieten von Natur aus ein gewisses Maß an Sicherheit, da sie als isolierte Prozesse laufen und unabhängig von anderen Containern arbeiten können.

Um das Risiko durch von Angriffen auf Container zu reduzieren, soll das Minimieren von Paketen und Abhängigkeiten für die jeweilige Anwendung geprüft werden. Dies hilft die Angriffsfläche weiter zu reduzieren, verringert Rauschen, ermöglicht schnellere Übersetzungszeiten innerhalb der CI/CD-Pipeline und minimiert die Dateigröße von Containern. Hierzu bieten sich vorgefertigte „Distroless“ Container (z.B. von Google[12], scratch[13], chiselled Ubuntu[14], usw.), reduzierte Linux-basierte Betriebssysteme (z.B. Alpine Linux[15]) oder Optimierungswerkzeuge (z.B. Docker Slim[16]) an.

3.2.1 Weitergabe von Containern an Dritte

Container, die von EuroDaT erzeugt, aber von Dritten bezogen und betrieben werden, gilt es mittels kryptographischer Schlüssel zu signieren. Somit können Dritte die Authentizität und Integrität von Containern prüfen, womit die Wahrscheinlichkeiten für Angriffe über die

---

[12] https://github.com/GoogleContainerTools/distroless

[13] https://hub.docker.com/_/scratch

[14] https://ubuntu.com/engage/chiselled-ubuntu-images-for-containers

[15] https://alpinelinux.org

[16] https://github.com/slimtoolkit/slim

Supply-Chain weiter reduziert werden.

Container sollen voneinander unabhängig betrieben werden. Dabei gilt zu beachten, dass jeder Container nur einen Dienst bereitstellt.

1.1.1 Hosten von Containern

Für den Betrieb von Containern muss eine Planung des Patch- und Änderungsmanagements stattfinden. Dabei wird festgehalten, wann und wie diese Aktivitäten durchgeführt werden.

Für Containern, die von Dritten bereitgestellt werden, allerdings von EuroDaT verwendet werden, wird zuerst die Authentizität und Integrität des Containers geprüft. Dazu ist zu dokumentieren, aus welcher Quelle ein Container stammt und ob die Quelle als vertrauenswürdig einzustufen ist. Für die Verwendung von Applikationen von extern in der EuroDaT-Plattform sollten signierte Container verwendet werden und die Signatur beim Hosten zu überprüfen. Nicht vertrauenswürdige Quellen gilt es auf Schwachstellen zu prüfen.

1.2 Softwaretests

In der Softwareentwicklung bei EuroDaT finden Tests statt. Wenn Testdaten für das Durchführen von Tests notwendig sind, dürfen keine Produktivdaten verwendet werden, die besonders schützenswerte oder personenbezogene Daten enthalten. Die Tests sind dabei in unterschiedlichen Kategorien einsortiert und decken verschiedene Testzwecke ab. Die Testkategorien bei EuroDaT bestehen aus den in der Softwareentwicklung üblichen Testkategorien Moduletests/Unittests, Integrationstest/End-to-End-Test, Systemtest, Abnahmetest. Die folgenden Abschnitte erläutern die verschiedenen Testkategorien.

1.2.1 Moduletests/Unittests

Nach der Erstellung der Software wird die technische Funktionsfähigkeit der einzelnen Module (auch Komponenten oder Units genannt) von der Entwicklerin auf der Entwicklungsumgebung geprüft. Ziel des Modultests ist es, eine technisch stabile und funktionierende Software für den Abnahmetest bereitzustellen.

Auftretende Fehler können im Anschluss an den Test eines Moduls direkt auf der Entwicklungsumgebung durchgeführt werden. Bei modular aufgebauter Software genügt anschließend im Rahmen des Modultests ein Test des betroffenen Moduls.

Bei der Verwendung von Continuous Integration (CI) (z.B. mit GitLab) sollten Testfälle für automatisierte Tests hinterlegt und automatisiert ausgeführt werden. Die Testergebnisse werden angezeigt („Ampelfarbe“) und können von den Entwicklerinnen ausgewertet werden. Die Testfälle für die einzelnen automatisierten Testdurchläufe werden dabei vom Git-Tool historisiert. Der durchgeführt Modultest wird anschließend in einem Protokoll festgehalten. Im Fall von CI befindet sich die Dokumentation und Auswertung bereits im Git-Tool.

1.2.2 Integrationstest

Besteht die Software aus mehreren Komponenten oder sind weitere Systeme beteiligt, so ist

die Software als Einheit aller Komponenten von den Entwicklern zu testen. Im Gegensatz zum Modultest, bei dem nur die einzelnen Module getestet wurden, wird hier die Funktionsfähigkeit der integrierten Software getestet.

Der Test wird auf der Entwicklungsumgebung durchgeführt. Nach erfolgtem Test können die aufgetretenen Fehler direkt von den Entwicklerinnen behoben und die Software anschließend noch einmal komplett integriert getestet werden.

Ist die Software nicht in mehrere Module unterteilt und werden nicht weitere Systeme angebunden, so entspricht der Integrationstest dem Modultest, so dass die zweistufige Unterteilung entfallen kann.

Auch hier kann die Durchführung automatisierter Tests durch ein Git-Tool in seinem Aufwand reduziert und in der Zuverlässigkeit erhöht werden. Da nach erfolgtem Bugfixing immer ein kompletter Re-Test erfolgt, können automatisierte Tests den Aufwand erheblich reduzieren bei gleichzeitiger Erhöhung der Sicherheit. Die Tests werden ebenfalls automatisiert im Git-Tool ausgewertet. Der durchgeführte Modultest wird anschließend in einem Protokoll festgehalten. Im Fall von CI befindet sich die Dokumentation und Auswertung bereits im Git-Tool.

1.2.3 Systemtests

Während in den niedrigeren Teststufen zunächst nur die technischen Spezifikationen des integrierten Systems überprüft wurden, werden im Systemtest (auch als End-to-End-Test bzw. E2E-Test bezeichnet) die Anforderungen der Fachspezifikation anhand der im Testkonzept dargestellten Testfälle überprüft.

Die Systemtests sollten an einem Setup stattfinden, welches ein möglichst nahes Setup zur Produktionsumgebung darstellen. Bei Verwendung einer Virtualisierung durch Container sollten die Systemparameter und Einstellungen möglichst exakt dem produktiven Betrieb entsprechen. Für den Systemtest dürfen die Entwickler noch auf das Testsystem zugreifen, um bei Fehlern schnelle Analysen und Bugfixes durchführen zu können.

Zusätzlich ist auch das Berechtigungskonzept für die User umzusetzen, so dass die Tester unter ihren produktiven Berechtigungen testen können. Eine falsche Vergabe der Rechte kann zu abweichenden Ergebnissen der Test führen. Insbesondere können Testfälle zur Informationssicherheit ohne korrekte Rechtevergabe nicht durchgeführt werden.

Der durchgeführt Systemtest wird anschließend in einem Protokoll festgehalten.

1.2.4 Abnahmetests

Nach erfolgreichem Systemtest wird der Abnahmetest ebenfalls auf einer frisch installierten Abnahmeumgebung durchgeführt. Die Installation erfolgt dabei mit den Paketen analog der späteren Produktivnahmen.

Wie beim Systemtest werden die Testfälle des Testkonzepts von den Testerinnen, die zuvor von der Owner der Software bestimmt wurden, durchgeführt. Die Testerinnen erhalten für die

Testdurchführung analog dem Systemtest die produktiven Berechtigungen. Beim Abnahmetest werden die Entwicklerinnen vom Zugriff auf das Abnahmesystem ausgeschlossen, da so ein produktiver Betrieb simuliert werden kann.

Bei auftretenden Fehlern wird ein Bugfixing mit anschließender Wiederholung des gesamten Tests durchgeführt.

Nach erfolgreichem Abnahmetest ist von der Owner ein Testprotokoll zu erstellen und die Testdaten abzusichern. Anschließend wird noch ein Abnahmeformular von der Owner ausgefüllt und unterschrieben. Anschließend kann die Software auf dem produktiven System installiert werden. Die Freigabe muss zudem in einem Ticket dokumentiert werden, z.B. indem die Mail-Freigabe dort hinterlegt wird.

1.1.1 Anmerkungen zum Bugfixing

Bei der Entwicklung kommen geeignete Verfahren zum Reduzieren von Fehlern während der Entwicklung zum Tragen. Empfohlen ist der Ansatz des Test-Driven Developments (TDD). Hierbei beschäftigen sich Entwicklerinnen frühzeitig mit möglichen Fehlerfällen und neue Fehler durch Änderungen können schneller erkannt werden. Bei neu auftretenden Fehlern, die noch nicht mittels Tests erfasst sind, werden Tests, die den Fehler adressieren, hinzugefügt.

Auch ohne Einsatz von TDD sollte jedes Bugfixing eine erneute Durchführung von Testzyklen beinhalten. Werden Daten extrahiert, können zuvor auch Regressionstests durchgeführt werden, um bereits vor Testdurchführung mögliche Seiteneffekte bisher positiv getesteter Daten ermitteln zu können. Bei Datenausleitungen kann ein Regressionstest auch für eine Abnahme herangezogen werden, wenn sich die erwarteten Ergebnisse gemäß Testkonzept darstellen lassen.

Darüber hinaus können beim Deployment, falls möglich, auch automatisierte Tests durchgeführt werden. Falls ein solcher Continuous-Integration-Ansatz möglich ist (z.B. bei GitLab), ist dies ein wichtiger Schritt zur Erleichterung der Re-Tests nach dem Bugfixing.

1.2 Dokumentation

Während des Entwicklungsprozesses ist auf eine angemessene Dokumentation zu achten. Dokumentation für neue und geänderte Funktionalitäten sollten gleichzeitig mit der Codeänderung in die Dokumentation übernommen werden. Die Art der Dokumentation kann innerhalb des Codes als Kommentare und außerhalb des Codes in Form von Verschriftlichungen erfolgen. Das exakte Format der Dokumentation sollte innerhalb des Softwareprojektes festgehalten werden (beispielsweise in einem Wiki innerhalb des Gitlab-Projekts). Die Dokumentation soll ausreichende Projekt-, Funktions- und Schnittstellendokumentation enthalten, sodass ein Fachexperte mithilfe der Dokumentation den Quellcode nachvollziehen und weiterentwickeln kann. Die Dokumentation sollte dabei auch die Software-Architektur und Bedrohungsmodellierung umfassen. Die Betriebsdokumentation sollte, als Teil der Gesamtdokumentation, konkrete Sicherheitshinweise für die Installation und Konfiguration für Administratoren, sowie für die

Benutzung des Produkts beinhalten. Weiterhin ist der Inhalt der Dokumentation in regelmäßigen Abständen auf seine Korrektheit zu untersuchen. Dies gilt insbesondere nach der Einführung von größeren Änderungen innerhalb der Software.

2 Prozess der Softwareentwicklung

Software bei EuroDaT wird nach der Scrum-Methode entwickelt. Dabei dienen die Software-Management-Tools (z.B. JIRA und Gitlab) als Hilfsmittel, um den Projektfortschritt zu planen und festzuhalten. Genauere Informationen für das erfolgreiche Umsetzen der Scrum-Methode finden sich im Scrum-Guide[17].

[17] https://scrumguides.org/

3 Verwaltung von Quellcode

Von EuroDaT entwickelter Quellcode wird in Gitlab verwaltet. Die Tools von Gitlab sollten zum Einrichten von Pipelines verwendet werden. Gitlab gilt auch als zentrales Backup von Quellcode und zum Erstellen von Container-Images für Releases. Für jedes Softwareprodukt von EuroDaT sollte ein eigenes Repository eingerichtet werden. In Gitlab sollten privilegierte Zugänge nach dem Least-Privilege-Prinzip vergeben werden (siehe dazu EuroDaTs Richtlinie für Identitäts- und Zugriffsmanagement).