Kooperationsrichtlinien

(1) Dieses Dokument präzisiert die allgemeinen Richtlinien für unsere Zusammenarbeit, deren spezifische Optionen im *Entwicklungs*- oder *Wartungsauftrag* vereinbart werden, der sich auf dieses Dokument bezieht.

(2) Kooperationspartner sind OÜ Rumma & Ko als **Entwickler** und der Auftraggeber als **Nutzer** eines *Gesamtsystems*, das die Endbenutzer als „ihren Lino" verstehen.

(3) Dieses Dokument ist öffentlich und gilt für alle Aufträge, die sich darauf beziehen. Die jeweils aktuelle Version steht unter https://saffre-rumma.net/dl/Kooperationsrichtlinien.pdf

(4) Der *Entwickler* kann dieses Dokument jederzeit anpassen und informiert dann den *Nutzer* über die Änderung. Wenn der *Nutzer* innerhalb von zwei Wochen nach Mitteilung einer Änderung keinen Einspruch erhebt, gilt die Änderung als akzeptiert. Solange eine Änderung in diesem Dokument nicht mitgeteilt und akzeptiert wurde, gilt die am Datum der Auftragserteilung gültige Version.

Systemkomponenten

(5) Das Gesamtsystem besteht aus *Hardware, Systemsoftware, Anwendungssoftware,* lokalen Konfigurationsdateien und *Dokumentation*.

(6) Die **Hardware** besteht aus dem **Server**, einem eigenständigen Rechner innerhalb oder außerhalb der Gebäude des Nutzers, den **Clients**, d.h. den Rechnern der Endbenutzer, sowie dem **Netzwerk**, das die Verbindung der beiden ermöglicht.

(7) Unter **Anwendungssoftware** verstehen wir die im Auftrag benannte Lino-Anwendung, die der *Entwickler* entwickelt und als freie Software öffentlich zur Verfügung stellt in Form von einem oder mehreren code repositories.

(8) Unter **Systemsoftware** verstehen wir alle Software oberhalb von Lino. Dazu gehören das Betriebssystem sowie Systemdienste wie Web-Server, Datenbank-Server, Mail-Server, File-Server, cron, logrotate, supervisor. Wir verwenden üblicherweise *Debian* als Betriebssystem, *Apache* als Web-Server, *MySQL* oder *PostgreSQL* als Datenbank-Server, *Postfix* als Mail-Server.

(9) Eine **Anlage** („Site") ist, wenn eine konkrete Version der *Anwendungssoftware* auf einem konkreten *Server* läuft. Sie besteht aus einem Domainname, hinter dem ein Webserver läuft, der Datenbank sowie den lokalen Einstellungen und Konfigurationsdateien zur Einbindung der *Anwendungssoftware* in die *Systemsoftware*.

(10) Auf einem *Server* können eine oder mehrere *Anlagen* laufen.

(11) Für die Bereitstellung und Wartung der *Hardware* sowie Backups aller lokalen Daten ist der **Systemverwalter** des *Nutzers* verantwortlich. Diese Aufgabe kann der Nutzer an einen externen Dienstleistungsanbieter delegieren.

(12) Der *Entwickler* ist verantwortlich für die Einrichtung und Wartung der *Software* auf dem *Server*. Zu diesem Zweck designiert er einen oder mehrere *Anlageverwalter.*

Arten der Zusammenarbeit

(13) Wir unterscheiden zwei grundlegende Arten der Zusammenarbeit : **Neuentwicklungsprojekte** und **Produktionsbetrieb**. *Im Produktionsbetrieb können gegebenenfalls* **Weiterentwicklungsprojekte** *gestartet werden.*

(14) Ein *Neuentwicklungsprojekt* wird durch einen **Entwicklungsauftrag** gestartet und endet, wenn die Anlage in den *Produktionsbetrieb* wechselt.

(15) Ziel eines *Neuentwicklungsprojektes* ist die Inbetriebnahme einer neuen *Anlage*, auf der die *Software* läuft und benutzt werden kann. Bei einem *Neuentwicklungsprojekt* trägt der *Nutzer* das Risiko der Kostenplanung.

(16) Ziel eines *Wartungsauftrags* ist die Aufrechterhaltung eines zuverlässigen *Produktionsbetriebs* einer *Anlage*.

Neuentwicklungsprojekte

(17) Bei einem *Neuentwicklungsprojekt* durchläuft unsere Zusammenarbeit üblicherweise folgende Phasen und Entwicklungsschritte:

(18) Der Entwickler trifft sich mit dem Nutzer zu einem **Interview**. Ziel des Interviews ist es, eine Übersicht über die Funktionen.

(19) Der Entwickler setzt einen **Entwicklungsauftrag** auf, in dem die beim Interview besprochenen Funktionen beschrieben sind sowie eine Schätzung der Entwicklungskosten.

(20) Der *Nutzer* unterschreibt den *Entwicklungsauftrag*.

(21) Zunächst eine **interne Entwicklungsphase** mit Analyse, gelegentlichen Fragen an die Kontaktperson, Programmierung der ersten Arbeitsversion und Einrichten eines *Prototypen*.

(22) Die *Vorbereitungsphase* endet, wenn der *Entwickler* den *Prototypen* als zufriedenstellend erachtet. Er teilt dem dem Nutzer die Zugriffsdaten zum *Prototypen* mit und schreibt eine Rechnung über Einrichtung des Prototypen und Ankauf von Stundenkredit für die Weiterentwicklung.

(23) Nach Zahlung der Rechnung startet die **gemeinsame Entwicklungsphase**, in der sich die folgenden Etappen zyklisch wiederholen : *Baustellenbesichtigung* mit Kontaktperson, Bearbeitung der festgestellten Probleme, Entwicklung und Installation einer neue Version und Planung der nächsten *Baustellenbesichtigung*. Wir arbeiten während dieser Phase auf dem *Prototypen*, der sich ständig verändert und weiterentwickelt.

(24) Am Ende der gemeinsamen Entwicklungsphase ist die Anwendung so weit fertig, dass der **Vorproduktionsbetrieb** beginnen kann. Die fiktiven Testdaten werden gelöscht, die Konfigurationsdaten angepasst. Eventuell werden Daten importiert aus bestehenden Quellen. Die Datenbank wird ab jetzt bei Aktualisierungen nicht mehr gelöscht, sondern migriert. Der *Prototyp* wird zu einer produktiven Anlage. Die *Kontaktperson* organisiert Schulung der Endbenutzer und Erfassung der Stammdaten. Probleme, die sich bei der täglichen Arbeit ergeben, werden erfasst, analysiert und behoben.

(25) Das Entwicklungsprojekt gilt als beendet, wenn *Entwickler* und *Nutzer* sich einig sind, dass das Ziel des Entwicklungsprojekts erreicht ist und man nun einen Wartungsauftrag starten kann.

(26) Die Dauer jeder Phase hängt von vereinbarten Terminen, Verfügbarkeit der *Kontaktperson* und den sich ergebenden Fragestellungen ab.

Produktionsbetrieb

(27) Als Anlage im **Produktionsbetrieb**, bezeichnen wir eine Anlage, für die ein **Wartungsauftrag** besteht.

(28) Im *Wartungsauftrag* übernimmt der *Entwickler* die technische Verantwortung für das zuverlässige Funktionieren der *Anlage*.

(29) Bei einem *Wartungsauftrag* zahlt der *Nutzer* einen vereinbarten **Jahresbeitrag**, der sich zusammensetzt aus (a) einem Beitrag zur *allgemeinen Entwicklung des Frameworks* und (b) einer vorgesehenen Anzahl von Stunden für *Dienstleistungen des Entwicklers*. (sh. *Stundenkontingent)*

(30) Ein *Wartungsauftrag* gilt für eine festgelegte Periode. Die Dauer eines Wartungsauftrags beträgt üblicherweise ein Jahr.

(31) Wartungsaufträge können formlos verlängert werden, indem der *Entwickler* eine Rechnung für die nächste Wartungsperiode ausstellt. Mit der Zahlung der Rechnung erklärt der *Nutzer* sein Einverständnis zur Verlängerung.

(32) Eventuelle Absicht zur Nicht-Verlängerung eines Wartungsauftrags wird dem Partner unverzüglich mitgeteilt.

(33) Vereinbarte Preise werden jedes Jahr entsprechend des belgischen Gesundheitsindexes angepasst.

(34) Entwickler und Benutzer können jederzeit einvernehmlich die Höhe des Jahresbeitrags anpassen.

(35) Rechnungen für Wartungsaufträge dürfen nach schriftlicher Absprache auch stückweise

bezahlt werden, also z.B. ¼ der Gesamtsumme alle drei Monate. Wobei der *Nutzer* dann selbst dafür verantwortlich ist, dass jede einzelne Teilzahlung rechtzeitig ausgeführt wird.

Weiterentwicklungsprojekte

(36) Ein **Weiterentwicklungsprojekt** *ist ein Entwicklungsprojekt, das* parallel mit dem Wartungsauftrag auf einer Anlage im Produktionsbetrieb läuft.

(37) Für Weiterentwicklungsprojekte erstellt der Entwickler eine Aufwandsprognose, die der Nutzer zum Ankauf weiterer Kontingentstunden berücksichtigt.

Ablauf eines Lino-Projekts

Schritt	***Endet durch***
Interview mit dem Nutzer. Analysegespräch.	Nutzer unterschreibt Entwicklungsauftrag.
Interne Entwicklungsphase zur Vorbereitung des Prototyps.	Zugriffsdaten zum *Prototyp* + Rechnung für Einrichten des *Prototypen* und Ankauf von *Entwicklungskontingent*.
Gemeinsame Entwicklungsphase	Beschluss des Nutzers*, in den Vorproduktionsbetrieb zu wechseln*
Vorproduktionsbetrieb	Abnahme. Nutzer und E sind sich einig, dass das E-Ziel erreicht ist. Einigung *Wartungskontingent* und *Wartungsauftrag*
Produktionsbetrieb	Ende des Wartungsauftrags

Kommunikation

(38) Der Nutzer stellt eine **Kontaktperson** zur Verfügung, die unsere Zusammenarbeit koordiniert, den Informationsfluss zwischen Entwickler und *Endbenutzern* zentralisiert und für die Endbenutzer je nach Bedarf Support, Dokumentation und Schulung organisiert.

(39) Eine E-Mail gilt als schriftliche Mitteilung, wenn ihr Eingang vom Empfänger bestätigt wurde.

(40) Der Nutzer muss eventuelle Probleme dem Entwickler melden, Rückfragen des Entwicklers beantworten und ihm alle nötigen Informationen liefern, die er zur Lösungsfindung braucht.

(41) Der Nutzer muss mindestens per E-Mail erreichbar sein, idealerweise auch telefonisch, über *instant messaging* oder Internet-Telefon

(42) Der Entwickler ist werktags von 8 bis 18 Uhr telefonisch **erreichbar**.

(43) Der Entwickler beantwortet jede Problemmeldung des Nutzers innerhalb einer entsprechend der Dringlichkeit des Problems vertretbaren Zeit.

(44) Nach vorheriger Absprache garantiert der Entwickler **erhöhte Bereitschaft**.

(45) Wenn der Benutzer einen Einsatz des Entwicklers vor Ort wünscht, kontaktiert er diesen zwecks Terminabsprache.

(46) Der *Anlageverwalter* teilt dem *Systemverwalter* alle Informationen mit, die dieser wünscht, um notfalls auch ohne Hilfe des *Anlageverwalters* Einsätze auf der *Anlage* durchzuführen.

(47) Der *Systemverwalter* teilt dem *Anlageverwalter* alle Informationen über Änderungen an der Hardware mit.

Stundenkontingente

(48) Ein **Stundenkontingent** ist ein Konto, das durch Ankauf von Stunden kreditiert und durch die Dienstleistungen des *Entwicklers* debitiert wird.

(49) Der Entwickler schreibt monatliche **Dienstleistungsberichte** über die von ihm geleisteten Arbeiten, die das Stundenkontingent kreditieren. Je nach Aktivität können Dienstleistungsberichte auch häufiger oder seltener als monatlich verschickt werden.

(50) Ein *Stundenkontingent* kann jederzeit durch Ankauf weiterer Stunden kreditiert werden.

(51) Beim Ankauf von Stundenkredit wird eine **vorgesehene Periode** festgelegt. Sollte der Kredit nicht aufgebraucht werden innerhalb dieser Periode, kann die Periode auf Anfrage des *Nutzers* und nach Einverständnis des *Entwicklers* verlängert werden.

(52) Wenn ein Stundenkontingent **früher als vorgesehen** auf unter 10 Stunden fällt, schreibt der Entwickler dem Nutzer ein Angebot zum Ankauf zusätzlicher Stunden.

(53) Bei **negativem Stundenkontingent** arbeitet der Entwickler ausschließlich an Anfragen, für die der Nutzer schriftlich bestätigt hat, dass er die nötigen Stunden im Nachhinein kaufen wird.

Aktualisierungen („Upgrades“)

(54) Jede *Anlage* ist individuell auf die Bedürfnisse des Nutzers zugeschnitten und kann bei Änderungen der Bedürfnisse angepasst werden.

(55) Ein **Einsatz** ist ein vom *Anlageverwalter* durchgeführter Eingriff auf den *Server* des *Nutzers*, bei dem die Konfigurierung des Servers geändert und/oder die Anwendungssoftware aktualisiert wird.

(56) Bei einer **Aktualisierung** wird neuer Quellcode der *Anwendungssoftware* auf die *Anlage* geladen und aktiviert.

(57) Ein **Release** ist eine *Aktualisierung*, für die eine eigene Versionsnummer und eigene Release-Notizen erstellt und gewartet werden.

(58) Eine **einfache Aktualisierung** kann durchgeführt werden, um Probleme nach einem *Release* operativ zu beheben. Dabei wird keine neue Versionsnummer vergeben und keine neue Testphase durchgeführt, und die Release-Notizen der laufenden Version werden lediglich aktualisiert.

(59) Bei jeder *Aktualisierung* ist der Anlageverwalter verantwortlich für eine korrekte **Datenmigration** und sorgt dafür, dass der Einsatz falls nötig rückgängig gemacht werden kann.

(60) Lino enthält eine umfangreiche **Testsuite** zum automatisierten Testen vieler Funktionen. Die Testsuite wird automatisiert vor jedem Release durchgeführt.

(61) Die *Testsuite* kann keine absolute Sicherheit vor Problemen nach einem *Einsatz* geben.

(62) Eine **Regression** ist, wenn etwas, das vor einer Aktualisierung funktionierte und nachher nicht mehr.

(63) Eine **Nebenwirkung** ist etwas, das nach einer Aktualisierung anders funktioniert als zuvor und deshalb potentiell Verwirrung oder Schulungsbedarf schafft.

(64) Wir bieten zwei unterschiedliche Verfahren, um mit dieser Art von Problemen umzugehen. Im Testverfahren **Flex** sind **Flexibilität** und **Aufwand** prioritär, im Aktualisierungsverfahren **Safe** dagegen eher **Planbarkeit** und **Dokumentation**.

Einfaches Testverfahren (Flex)

(65) Der Vorteil des **einfachen Testverfahrens** besteht darin, dass es kostengünstig und flexibel ist.

(66) Bei diesem Verfahren werden neue Versionen nach internem Test durch den Entwickler über Nacht aufgespielt.

(67) Am nächsten Arbeitstag vereinbaren die *Endbenutzer* vor dem Einsatz eine **Testphase**, die eine oder mehrere Stunden dauern kann.

(68) In dieser Zeit testen ausgewählte Benutzer ihren neuen Lino im laufenden Produktionsbetrieb.

(69) Sie sind dabei auf Regressionen und Überraschungen gefasst, die sie dann melden müssen und die dann prioritär bearbeitet werden.

(70) Alle sind sich dessen bewusst, dass der Einsatz rückgängig gemacht und die Anlage auf den Zustand vom Vortag zurück gesetzt werden könnte, falls ein Release-Blocker gemeldet wird.

Erweitertes Testverfahren (Safe)

(71) Beim **erweiterten Testverfahren** arbeiten wir mit einer **Vorschau** auf die kommende Version, die von ausgewählten Benutzern gründlich getestet wird, bevor sie in Produktion geht.

(72) Eine **Vorschau** ist eine zusätzliche Anlage neben der Produktionsanlage, üblicherweise auf dem gleichen Server, der eine Kopie der Produktionsdaten zeigt, wie sie mit der nächsten Version der Software aussähen.

(73) Der Vorteil des *erweiterten Testverfahrens* besteht darin, dass es reibungslosen Produktionsbetrieb ohne Unterbrechung gewährleistet.

(74) Die Testphase dauert so lange wie nötig.

(75) Auf Wunsch des Nutzers schreibt und wartet der Entwickler detaillierte Testanweisungen für die Tester.

(76) Wenn das Testen einer neuen Version begonnen hat, kommen keine neuen Features mehr hinzu.

(77) Ohne Benutzerdokumentation gibt es keine formalen Testanweisungen, sondern die Benutzer testen intuitiv bzw. nach selbst erstellten Checklisten.

(78) Das Umschalten der *Vorschau* in den *Produktionsbetrieb* findet statt, wenn alle Probleme in der *Vorschau* behoben wurden.

Dokumentation

(79) Der *Entwickler* schreibt bei jedem Release **Release-Notizen** in der vereinbarten Sprache (Deutsch, Französisch oder Englisch).

(80) Für jede *Anlage* im *Produktionsbetrieb* veröffentlicht und wartet der *Entwickler* **technische Dokumentation** über die *Anwendungssoftware* in englischer Sprache.

(81) Die *technische Dokumentation* umfasst standardmäßig eine **Spezifikation** aller Funktionen, deren Zielgruppe andere Entwickler sind und die ein integraler Bestandteil der *Testsuite* ist.

(82) Auf Wunsch des Nutzers schreibt der Entwickler ein **Benutzerhandbuch** und passt es bei Änderungen der Software an.

(83) Umfang und Qualität aller Dokumentation richten sich nach den Bedürfnissen des Benutzers. Der Entwickler arbeitet Verbesserungsvorschläge des Nutzers in das Handbuch ein als Teil der Wartungsarbeit.

(84) Das *Benutzerhandbuch* ist online verfügbar und kann optional als pdf-Datei heruntergeladen werden.

(85) Der Entwickler veröffentlicht das *Benutzerhandbuch* sowie den zur Erstellung nötigen Quellcode unter einer freien Lizenz.

Autorenrecht und Datenschutz

(86) Alle Mitarbeiter des Entwicklers, die Zugriff auf den Server oder Kontakt mit Endbenutzern haben, unterliegen den Datenschutzbestimmungen, die der Nutzer an seine eigenen Mitarbeiter stellt.

(87) Der Nutzer ist dafür verantwortlich, den Entwickler über seine Datenschutzbestimmungen zu informieren.

(88) Der Entwickler ist dafür verantwortlich, seine Mitarbeiter an die Datenschutzbestimmungen

des Nutzers zu binden.

(89) Der Entwickler ist verpflichtet, den im Rahmen der Zusammenarbeit erstellen Quellcode und Dokumentation der Software öffentlich unter einer freien Lizenz zur Verfügung zu stellen entsprechend der Anforderungen der Initiative *Public Money? Public Code!* (https://publiccode.eu/de/openletter/)

(90) **Betriebsabläufe** und **Vornamen** von *Kontaktpersonen* gelten nicht als vertraulich und dürfen in Quellcode und Dokumentation veröffentlicht werden.

(91) **Lokale** Konfigurationsdateien auf dem Server gelten als vertraulich und dürfen nicht veröffentlicht werden.

Haftungsausschluss

(92) Der Entwickler garantiert nicht die Fehlerfreiheit der Software, sondern seine Hilfestellung bei Problemen.

(93) Sollte jemand aufgrund eines Fehlers oder Ausfalls der *Anlage* zu Schaden kommen, kann der Entwickler dafür nicht rechtlich haftbar gemacht werden.

Glossar

(94) Als **Endbenutzer** bezeichnen wir alle Personen, denen der Nutzer Zugriff auf die Anlage gewährt. Der Nutzer ist verantwortlich für das Vergeben und Zurücknehmen von Zugriffsrechten.

(95) Zur **allgemeinen Entwicklung** der *Software* gehören z.B. interne Verbesserungen, technische Dokumentation, Anpassungen an neue Technologien, Entwicklung neuer Benutzerschnittstellen, Veröffentlichung des Quellcodes als Freie Software, Qualitätssicherung und Änderungen von allgemeinem Interesse in anderen Entwicklungsprojekten.

(96) Als **Prototyp** bezeichnen wir eine *Anlage* mit fiktiven Demo-Daten, die als Arbeitsgrundlage für die *aktive Entwicklungsphase* dient. Ein *Prototyp* kann entweder auf einem vom Entwickler betriebenem *Server* installiert sein, oder direkt auf dem *Server*, auf dem die *Software* später im *Produktionsbetrieb* laufen wird.

(97) Als **Wartungsarbeiten** gelten Antwort auf Fragen des *Nutzers* zur Benutzung der Software, Hilfestellung bei Diagnose von Problemen, *Analyse*, kleine Optimierungen der Software, für die sich ein *Weiterentwicklungsauftrag* nicht lohnt, *Einsätze* auf dem Server sowie Behebung von Problemen, die durch einen *Einsatz* verursacht wurden.

(98) Als **Analyse** bezeichnen wir die Diagnose und Erfassung eines Problems mit dem Nutzer, Suche nach Ursachen, Erarbeiten einer Lösung.

(99) Als **Qualitätssicherung** bezeichnen wir das Schreiben und Pflegen von automatisierten Testverfahren und Testanweisungen für manuelle Tests.

(100) Als **Deployment** bezeichnen wir die Veröffentlichung der Software mit dem Ziel, sie auch für andere Entwickler verfügbar zu machen.