D04-QM Programmierrichtlinien zur Erstellung von „PersoApp“-Softwaremodulen

Editor: Siniša Đukanović (Fraunhofer SIT)

Prüfung: Sven Wohlgemuth (TU Darmstadt/CASED)

Typ: [TECHNISCHER BERICHT]

Projekt: „PersoApp“

Version: 1.0

Datum: 19. Juni 2013

Status: [FREIGABE]

Klasse: [ÖFFENTLICHKEIT]

Datei: D04_QM_Programmierrichtlinien zur Erstellung von PersoApp-Softwaremodulen.docx

Zusammenfassung

Die vorliegende Programmierrichtlinie umfasst unter anderem ein Regelwerk für die Festlegung einer einheitlichen Syntax, Nomenklatur, Struktur, den Umfang und gegebenenfalls die einzusetzenden Verfahren, die bei der Erstellung und Wartung von Quelltext der „PersoApp“ zu berücksichtigen sind. Darüber hinaus wird durch sie festgelegt, in welcher Form externe Bibliotheken vor einer Verwendung in der „PersoApp“ zu prüfen sind. Das Ziel bei der Festlegung dieser Programmierrichtlinie ist letzten Endes einen Beitrag zur Sicherstellung der Qualität der Software über den Projektverlauf zu leisten.

Konsortialleitung:
Prof. Dr. Ahmad-Reza Sadeghi und Dr. Sven Wohlgemuth
System Security Lab, TU Darmstadt/CASED, Mornewegstr. 32, 64293 Darmstadt
Tel.: +49-6151-16-75561 Fax: +49-6151-16-72135
E-Mail: persoapp@trust.cased.de Web: https://www.persoapp.de

Nutzungslizenz

Die Nutzungslizenz dieses Dokumentes ist die Creative Commons Nutzungslizenz „Attribution-ShareAlike 3.0 Unported“[1].

Mitglieder des Konsortiums

1. **AGETO Service GmbH**, Deutschland
2. **Center for Advanced Security Research Darmstadt (CASED)**, Deutschland
3. **Fraunhofer Institut für Sichere Informationstechnologie (SIT)**, Deutschland
4. **Technische Universität (TU) Darmstadt**, Deutschland

Versionen

Version	*Datum*	*Beschreibung (Editor)*
0.1	2013-05-28	Initiale Erstellung (Siniša Đukanović)
0.2	2013-06-17	Überarbeitete Version (Siniša Đukanović)
0.3	2013-06-17	Reviewversion (Sven Wohlgemuth)
1.0	2013-06-19	Freigabeversion (Siniša Đukanović)

Autoren

Autoren	*Beiträge*
Philipp Holzinger (Fraunhofer SIT)	Kapitel 1 - 4, 5, 5.1, 5.3, 6 - 8
Stefan Triller (Fraunhofer SIT)	Kapitel 1 - 4, 5, 5.1, 5.3, 6 - 8
AGETO	Kapitel 5.2, 5.4 - 5.11

[1] http://creativecommons.org/licenses/by-sa/3.0/

Inhaltsverzeichnis

1 Ziel und Zweck des Dokumentes

Die Software-Entwicklung im Rahmen dieses Open-Source-Projekts „PersoApp“ erfolgt durch eine aktive Community, deren Mitglieder aus dem privaten, behördlichen oder industriellen Umfeld kommen. Diese Heterogenität hat zur Folge, dass es grundlegend unterschiedliche Auffassungen, Bedürfnisse und Möglichkeiten innerhalb der Entwicklergemeinschaft gibt, wie Quelltext zu erstellen, wiederzuverwenden und abzulegen ist. Um einen gemeinsamen Leitfaden für die Implementierung sicherer und qualitativ hochwertiger Softwarekomponenten für die PersoApp bereitzustellen, wurde das vorliegende Dokument „D04-QM Programmierrichtlinien zur Erstellung von „PersoApp“-Softwaremodulen“ ausgearbeitet.

Die vorliegende Programmierrichtlinie umfasst unter anderem ein Regelwerk für die Festlegung einer einheitlichen Syntax, Nomenklatur, Struktur, den Umfang und gegebenenfalls die einzusetzenden Verfahren, die bei der Erstellung und Wartung von Quelltext der PersoApp zu berücksichtigen sind. Darüber hinaus wird durch sie festgelegt, in welcher Form externe Bibliotheken vor einer Verwendung in der „PersoApp“ zu prüfen sind.

Das Ziel bei der Festlegung dieser Programmierrichtlinie ist letzten Endes einen Beitrag zur Sicherstellung der Qualität der Software über den Projektverlauf zu leisten. Auf Dokument „D02-QM Qualitätskriterien: Aufbau, Messgrößen und Bewertung“ verweisend, betrifft dies insbesondere die folgenden Qualitätskriterien:

- Modularität
- Übertragbarkeit und Wiederverwendbarkeit
- Wartbarkeit
- Projektzugänglichkeit
- Verständlichkeit
- Integrationsfähigkeit

2 Anwendungsbereich

Die vorliegende Programmierrichtlinie adressiert die unterschiedlichen Auffassungen und Präferenzen innerhalb der heterogenen Entwicklergemeinschaft, indem sie ein gemeinsames und einheitliches Regelwerk für die Erstellung, Wartung und Wiederverwendung von Quelltext im Rahmen des Open-Source-Projekts „PersoApp“ zur Verfügung stellt. Daher sind die hier festgelegten Konventionen und Regeln von allen Projektteilnehmern verbindlich einzuhalten, um konsistent die Qualität der entwickelten Software positiv beeinflussen zu können.

3 Abkürzungen und Begriffsdefinition

Begriff	*Abkürzung*	*Definition*
CamelCase	---	CamelCase bezeichnet die Verwendung von Groß- und Kleinbuchstaben innerhalb eines Wortes.
Common Vulnerability Scoring System	CVSS	Bei CVSS handelt es sich um eine weit verbreitete Bewertungsmethode für Schwachstellen. Unter Berücksichtigung unterschiedlicher Charakteristika (den sog. Metrics) einer bestimmten Schwachstelle wird ein individueller Score berechnet. Mehrere derartig bewertete Schwachstellen können anhand ihrer individuellen Scores miteinander verglichen werden, um letztlich eine Priorisierung vorzunehmen, oder um als Grundlage für eine Risikobewertung zu dienen. Vgl. http://www.first.org/cvss/cvss-guide.pdf

Notiz: Der Titel des vorliegenden Dokumentes „D04-QM Programmierrichtlinien zur Erstellung von „PersoApp“-Softwaremodulen“ sowie die Überschrift aus Kapitel 5 sind dem Angebotsschreiben entsprechend übernommen worden. Tatsächlich handelt es sich hier aber um die Entwicklung von „*PersoApp“*-Softwaremodulen, die mit der „AusweisApp“ der OpenLimit SignCubes AG in keinem direkten Zusammenhang stehen. Im Fortlauf des Dokumentes wird daher der korrekte Name „PersoApp“ verwendet. Der Begriff „AusweisApp“ in diesem Dokument bezieht sich nur auf das Softwareprodukt der OpenLimit SignCubes AG, wenn dies explizit angegeben wurde.

4 Zuständigkeiten und Verantwortlichkeiten

Dieses Dokument wurde in gemeinschaftlicher Zusammenarbeit vom Fraunhofer-Institut für Sichere Informationstechnologie (SIT) und der AGETO Service GmbH erstellt. Die weiteren Konsortialpartner wurden in Form eines Reviews mit der Möglichkeit für Kommentare und Änderungswünsche indirekt daran beteiligt.

Die Anwendung der Programmierrichtlinie obliegt jedem Projektteilnehmer, der Softwaremodule für die „PersoApp“ entwickelt, wartet, erweitert oder im Zuge einer Wiederverwendung externe Bibliotheken in die Software einbindet.

5 Programmierrichtlinien zur Erstellung von PersoApp“-Softwaremodulen

Im Zuge dieser Programmierrichtlinie wird ein einheitliches und allgemein akzeptiertes Regelwerk ausgearbeitet, dass die Erstellung, Wartung und Wiederverwendung von Quelltext im Zuge des „PersoApp“-Projektes adressiert. Hierfür werden Festlegungen, unter anderem bezüglich der Syntax, Nomenklatur und Strukturierung getroffen, die im Folgenden detailliert dargestellt werden.

5.1 Allgemeine Richtlinien und Konventionen

Im Zuge der Erstellung der vorliegenden Programmierrichtlinie wurde durch eine sorgfältige Vorgehensweise sichergestellt, dass keine substantiellen Festlegungen ausgelassen wurden. Sollte es dennoch im Projektverlauf zu fraglichen und undefinierten Situationen kommen, die in dieser Richtlinie keine Berücksichtigung finden, dann wird an dieser Stelle auf die „Code Conventions for the Java Programming Language“ (vgl. http://www.oracle.com/technetwork/java/codeconv-138413.html) von Sun Microsystems verwiesen, die derzeit von Oracle zur Verfügung gestellt werden.

5.2 Einzuhaltende technische Richtlinien

Die Sicherheitsanforderungen an den eID-Client und der Open-Source-Software-Bibliothek der PersoApp entsprechen den technischen Richtlinien (TR) des Bundesamtes für Sicherheit in der Informationstechnik (BSI):

- BSI-TR-03112 (eCard API Framework)
- BSI-TR-03127 (Architektur elektronischer Personalausweis)
- BSI-TR-03128 (EAC-PKI’n für den elektronischen Personalausweis)
- BSI-TR-03130 (eID-Server)

Eine Einhaltung dieser Sicherheitsanforderungen durch einen eID-Client wird durch Handlungsempfehlungen des Projektes „PersoApp“ für die Etablierung und Dokumentation einer Sicherheitsarchitektur und eine sichere Integration von Open-Source-Software-Komponenten unterstützt. Insbesondere bei einer Integration von Software in andere IT-Systeme sollen sie Einhaltung dieser Anforderungen durch das Gesamtsystem unterstützen.[23]

[2] https://www.bsi.bund.de/

[3] http://www.persoapp.de/open-source-eid-client

5.3 Verwendung externer Bibliotheken

In der heutigen Softwareentwicklung wird ein Softwareprodukt nicht mehr von genau einer Firma komplett programmiert und anschließend betrieben, es ist vielmehr eine Orchestrierung mehrerer (externer) Bibliotheken um die gewünschte Funktion zu erhalten. Hat man früher als Softwarehersteller die Entwicklungsprozesse der Softwareprodukte komplett selbst in der Hand gehabt, so fließen heute auch die Arbeiten Dritter in die Produkte ein. Ein häufiges Anwendungsbeispiel hierfür ist die Verwendung freier Open-Source-Bibliotheken, deren Lizenz es auch erlaubt sie in kommerziellen Produkten einzusetzen.

Auch im Rahmen dieses Projektes werden für die PersoApp Open-Source-Bibliotheken mit eingebunden werden, z.B. eine Bibliothek für kryptografische Operationen. Um sicherzustellen, dass diese externen Bibliotheken den Qualitätsansprüchen der „PersoApp“ genügen, werden verschiedene Kandidaten evaluiert und dokumentiert warum welche Bibliothek zu welchem Zweck eingesetzt wird.

Um die Sicherheitsqualität externer Bibliotheken abschätzen zu können, ist es hilfreich die folgenden Fragen an (die Entwickler) der externen Bibliothek zu stellen:

Gibt es auf der Projektwebseite eine leicht zu findende Seite zum Thema Sicherheit?

Ein wichtiger Punkt beim Einsatz fremder Bibliotheken ist der allgemeine Umgang mit dem Thema Sicherheit seitens der Projektbeteiligten. Falls sie auf ihrer Projektwebseite noch nicht einmal etwas erklärenden Text über die Sicherheit in ihrer Software haben, ist es fraglich ob sie sich überhaupt Gedanken über das Thema Sicherheit gemacht haben. Sollten sie dennoch Sicherheit in ihr Produkt eingebaut haben, deutet der fehlende Hinweis auf der Webseite darauf hin, dass dies nur beiläufig ohne wirkliches Konzept erfolgt ist. Gibt es allerdings ein Sicherheitskonzept, welches nur den Projektbeteiligten bekannt ist, und fehlen daher Informationen zum Thema Sicherheit auf der Webseite, so sieht es eher nach „Security by obscurity“ aus und keiner kann transparent nachvollziehen, ob das Konzept schlüssig ist.

Gibt es auf der Projektwebseite eine Liste bekannter Schwachstellen und Sicherheitshinweise?

Viele Softwareprojekte veröffentlichen „security advisories“ zu behobenen Schwachstellen aus denen hervorgeht, in welcher Version sie enthalten waren und welche Auswirkungen sie hatten. Dies zeigt einen verantwortungsvollen Umgang mit dem Thema Sicherheit. Zu aktuellen Schwachstellen sollte ebenfalls auf der Projektwebseite Stellung genommen und den Nutzern Workarounds angeboten werden, was sie bis zur Behebung der Schwachstelle dagegen tun können.

Existiert ein Kontaktpunkt und Prozess um Schwachstellen zu melden auf der Webseite oder wird einer in der Dokumentation erwähnt? Entspricht es bewährten Verfahren?

Findet ein verantwortungsvoller Sicherheitsforscher eine Schwachstelle in einer Software, so möchte er die Projektbeteiligten zunächst darauf aufmerksam machen, damit sie die Schwachstelle beheben können. Er setzt den Projektbeteiligten hierzu eine angemessene Frist, bevor er die Schwachstelle veröffentlicht. Findet er keine Kontaktmöglichkeit, besteht die Gefahr dass er die Schwachstelle direkt veröffentlicht. Gängige Kontaktmöglichkeiten sind die Angabe einer E-Mail-Adresse oder ein Kontaktformular im Bereich „Sicherheit“ auf der Webseite, unter denen man die verantwortlichen Entwickler erreicht.

Hat das Projekt ein Bug-Tracking-System welches auch aktiv genutzt wird? Gibt es (offene oder geschlossene) Sicherheitsfehler in der aktuellen Version?

Jede Software beinhaltet Fehler und viele werden erst während der Benutzung der Software gefunden. Ein Bug-Tracking-System ermöglicht es den Entwicklern Fehlerbeschreibungen der Benutzer entgegenzunehmen und Fehler gegeneinander abzuwägen und zu priorisieren, welcher zuerst behoben wird. Um die Sicherheit eines Projektes beurteilen zu können, kann in solch einem System nach Hinweisen zu sicherheitsrelevanten Fehlern gesucht werden und man sieht ob sie bereits behoben wurden oder noch offen sind. Offene sicherheitsrelevante Fehler sollten noch nicht lange im System sein, sondern eher bevorzugt geschlossen werden.

Gibt es Schwachstellenberichte in öffentlichen Schwachstellendatenbanken wie z.B. der NIST National Vulnerability Database (NVD)?

Öffentliche Schwachstellendatenbanken listen sicherheitsrelevante Fehler bekannter Softwaresysteme. Eine Recherche in solchen Datenbanken kann Aufschluss darüber geben, ob es in der Vergangenheit solche Fehler in der Software gab. Findet man in den Datenbanken Fehler, die auf der Projektwebseite allerdings nicht erwähnt oder verlinkt sind, so deutet das darauf hin, dass das Projekt auf Sicherheit keinen großen Wert legt.

Falls es Schwachstellen im Bug-Tracking-System des Projektes oder einer Schwachstellendatenbank gab oder gibt:

Was waren die Auswirkungen dieser Schwachstellen (evtl. CVSS score)?

Berichte über Schwachstellen beinhalten oft eine Sektion über die möglichen Auswirkungen der Sicherheitslücke. Hatte die Software in der Vergangenheit eher Lücken mit schwerwiegenden Auswirkungen, die im Softwaredesign begründet sind, so deutet das darauf hin, dass Sicherheit nur am Rande betrachtet wurde. Sind die Auswirkungen durch Programmierfehler begründet, so deutet es auf eher unerfahrene Programmierer oder fehlende Reviews hin. Einen Anhaltspunkt, wie schwerwiegend eine Sicherheitslücke ist, kann ihr CVSS-Wert sein, sofern vorhanden.

Kann man auf einen Blick abschätzen wie komplex die Schwachstellen waren? Waren es einfache, vermeidbare Fehler, die durch das Befolgen von Secure Coding Practices verhindert werden hätten können, oder waren sie komplexer?

Wenn man sich den Quelltext an den Stellen anschaut, an denen Schwachstellen aufgetreten sind und den alten Quelltext mit dem neuen vergleicht, kann man einen

Einblick darüber bekommen, ob es eher vermeidbare oder komplexere Fehler waren. Flüchtigkeitsfehler, wie z.B. das Fehlen einer Längenüberprüfung bei Benutzereingaben, können durch Secure Coding Practices leicht verhindert werden. Komplexere Fehler, die darauf schließen lassen, dass Sicherheit gar nicht in der Architektur der Software vorgesehen war, kann man daran erkennen, dass größere Mengen an neuen Quelltextzeilen hinzugekommen sind um das Problem zu lösen.

Wie professionell hat das Projekt die Schwachstellen behoben? Die Zeit bis zur Behebung, als auch die Interaktion mit den Findern der Schwachstellen sind hier von Interesse.

Sofern die Schwachstellen des Projektes in dessen Bugtracking-System oder in einer öffentlichen Schwachstellendatenbank eingetragen sind, ist es interessant wie lange es gedauert hat bis sie geschlossen wurden. Öffentliche Mailinglistenarchive wie z.B. von Bugtraq oder Full-disclosure können ebenfalls nach dem Produkt durchsucht werden. Hier finden sich oft Anfragen von Sicherheitsforschern, die nicht wissen wie sie mit den Entwicklern eines Produktes in Kontakt treten können oder berichten darüber wie problematisch es war die Schwachstelle zu melden und geschlossen zu bekommen.

Falls die Bibliothek Sicherheitsfunktionen beinhaltet/anbietet:

Ist die Sicherheitsarchitektur gut dokumentiert?

Wenn das Produkt Sicherheitsfunktionen beinhaltet ist, sollten diese auch dokumentiert sein. Hierzu gehört u.a. die Sicherheitsarchitektur aus der hervorgehen sollte, welche Sicherheitsfunktionen in dem Produkt enthalten sind und wie sie zusammenhängen. Für jemanden, der das Produkt in seine eigenen Produkte integrieren möchte, sind insbesondere die Zusammenhänge und externen Anforderungen interessant. Als Beispiel sei die Bereitstellung von Verschlüsselungsalgorithmen genannt. Derjenige, der das Produkt in seine Produkte integriert, sollte nachvollziehen können, welche Zertifikate oder Schlüssel Verwendung finden und dass er Sorge dafür tragen muss diese „sicher“ zu verwalten.

Entsprechen die Sicherheitsfunktionen bewährten Verfahren? (z.B. gut bekannte und getestete kryptografische Algorithmen und Implementierungen falls Kryptografie zum Einsatz kommt, oder gibt es Unterstützung zur Durchsetzung von Passwortvorschriften?)

Sicherheit ist keine feststehende Eigenschaft, sondern ein Prozess, welcher gegen sich ändernde Bedrohungen evaluiert werden muss. Daher sollten eingesetzte Sicherheitsfunktionen dem „State-of-the-art“ entsprechen und nicht veraltet sein.

Gibt es eine Sicherheitsdokumentation für die Entwickler im Projekt? (Eine Suche nach Schlüsselwörtern aus dem Bereich „Sicherheit“ in der Entwicklerdokumentation oder im Quelltext ist hierbei hilfreich)

Insbesondere bei Open-Source-Projekten kommen immer wieder Entwickler mit unterschiedlicher Programmiererfahrung hinzu. Damit diese einen möglichst einfachen Einstieg in das Projekt bekommen, ist Dokumentation nötig. Die

Dokumentation zum Projekt sollte nicht nur beschreiben, wie ein neuer Entwickler den Quelltext kompiliert bekommt, sondern auch wie die Architektur der Software aufgebaut ist und welche Sicherheitsfunktionen in ihr integriert sind. Hinweise zur Verwendung „sicherer“ APIs sind ebenfalls von Vorteil.

Kann man schnell die Quelltextqualität abschätzen? (besserer Quelltext ist meistens sicherer) Ist der Quelltext ordentlich strukturiert? Enthält er Kommentare? Gibt es viele „TODO“ oder „FIXME“ Kommentare im Quelltext? Gibt es sinnvolle Kommentare bei Check-Ins im Versionskontrollsystem, die auch dem Quelltext entsprechen?

Open-Source-Projekte haben den Vorteil, dass der Quelltext öffentlich zur Verfügung steht. Wirft man einen Blick in den Quelltext und sucht nach den Stichworten „TODO“ oder „FIXME“ kann man einen schnellen Eindruck darüber gewinnen, wie sorgfältig die Programmierer gearbeitet haben. Oft steht ein „TODO“ an Stellen an denen der Entwickler schnell die Funktion sicherstellen wollte, sich aber durchaus bewusst ist, dass noch etwas (z.B. Sicherheit) fehlt. Im Versionskontrollsystem des Projektes kann man die Kommentare zu den Check-Ins nach IDs aus dem Bugtracking-System durchsuchen, um festzustellen ob geschlossene Lücken auch hier dokumentiert wurden.

Ist die Standardkonfiguration der Bibliothek sicher? Gibt es Standardpasswörter, vorkonfigurierte Verschlüsselungsalgorithmen etc.?

Wichtig für den Einsatz einer Bibliothek in einer anderen Software sind die externen Abhängigkeiten und Anforderungen, die erfüllt sein müssen damit die Bibliothek nicht nur funktioniert, sondern auch „sicher“ konfiguriert ist. Ein Blick in die Konfigurationsdateien gibt Aufschluss darüber ob z.B. Standardpasswörter verwendet werden, die unbedingt geändert werden sollten. Ebenso sollte nach optionalen Sicherheitsfunktionen geschaut werden, ob diese in der Standardkonfiguration aktiv sind.

Wurde die Bibliothek einer Sicherheitsprüfung unterzogen? Gibt es dazu öffentliche Ergebnisse?

Einige, meist populärere Bibliotheken wurden bereits Sicherheitsprüfungen, die von Sicherheitsexperten durchgeführt wurden, unterzogen. Interessant ist, ob es auf den Projektwebseiten Hinweise darauf gibt, und wenn ja, ob es öffentliche Ergebnisse gibt. Falls es Sicherheitsüberprüfungen gab, aber die Ergebnisse nicht veröffentlicht wurden, kann der Bug-Tracker des Projektes evtl. Hinweise auf geschlossene Lücken geben.

5.4 Quellcodeformatierung

Generelle Richtlinie für die Formatierung des Quellcodes sind die „Code Conventions for the Java Programming Language“.[4] Die Festlegung allgemeiner Richtlinien hilft, den Code auch bei einer Vielzahl unabhängig Beitragender lesbar zu halten. Gerade

[4] http://www.oracle.com/technetwork/java/codeconv-138413.html

bei Open-Source-Projekten kommen des Öfteren neue Entwickler hinzu und andere verlassen das Projekt. Somit ist es von großer Bedeutung, dass sich neue Entwickler schnell in den Code einarbeiten können. Neben einer guten Dokumentation ist ein guter, einheitlich formatierter Code ein wichtiger Bestandteil.

Untenstehend sind die Regeln zur Formatierung in Form einer Liste angegeben:

- Jede Datei (Klasse) muss mit einem Copyright-Kommentar beginnen, d.h., dass dieser insbesondere noch vor der Paketangabe platziert ist.
- Import-Anweisungen stehen direkt unter der Paketnamenangabe und sollen in sortierter, gruppierter Form angegeben werden.
- Namen für Pakete, Klassen, Typen, Methoden etc. werden der englischen Sprache entlehnt. Paketnamen bestehen ausschließlich aus Kleinbuchstaben (mypackage, de.mypackage etc.).
- Klassen-/Typnamen beginnen immer mit einem Großbuchstaben, wobei innerhalb des Namens in sinnvoller Art und Weise mehrfach Großbuchstaben vorkommen sollen (Panel, TransportProvider etc.).
- Methodennamen sind Verben und beginnen stets mit einem Kleinbuchstaben, wobei innerhalb des Namens in sinnvoller Art und Weise mehrfach Großbuchstaben vorkommen sollen (getName, isValid, compute).
- Variablennamen beginnen stets mit einem Kleinbuchstaben, wobei innerhalb des Namens in sinnvoller Art und Weise mehrfach Großbuchstaben vorkommen sollen (panel, transportProvider etc.).
- Namen für Konstanten (final-Variablen) dürfen nur aus Großbuchstaben und, zur Worttrennung, dem Unterstrich (_) bestehen.
- Einrückungen sollen durch 4 Leerzeichen (keine Tabs) pro Level erfolgen.
- Geschweifte Klammern für Codeblöcke sollen auch im Fall von Einzeilern verwendet werden. Die öffnende geschweifte Klammer ist mit einem Leerzeichen Abstand in dieselbe Zeile zu schreiben.
- Die maximale Zeilenbreite wird auf 160 Zeichen festgelegt.
- Codestellen, die noch Fehler enthalten werden mit /* FIXME <Kommentar> */ markiert.

5.5 Namensgebung

Die Dateinamen für den Quellcode besitzen die Endung .java und Bytecode trägt die Endung .class. Neben dem eigentlichen Quellcode kann das Archiv noch Anleitungen enthalten. Es wird empfohlen, diese mit README, INSTALL zu bezeichnen. Eine Datei namens LICENSE enthält den Lizenztext der für den Code anwendbar ist.

Innerhalb einer Java-Datei befindet sich eine einzelne Klasse bzw. ein Interface. Private Klassen und Interfaces können in derselben Datei wie die öffentliche (public) Klasse bzw. Interface gespeichert werden.

Namen für Pakete, Klassen, Typen, Methoden etc. werden der englischen Sprache entlehnt. Deutsche Bezeichnungen sind nach Möglichkeit zu vermeiden.

Namen von Paketen, wie beispielsweise mypackage oder de.mypackage, bestehen ausschließlich aus Kleinbuchstaben.

Namen von Klassen und Typen beginnen immer mit einem Großbuchstaben. Innerhalb einer Bezeichnung sollte CamelCase verwendet werden (Panel, TransportProvider etc.).

Die Namen der Methoden sind Verben, die eventuell mit weiteren Worten gekoppelt sind. Für solche Zusammensetzungen ist CamelCase zu verwenden (getName, compute etc.).

Der Namen von Variablen beginnt immer mit einem Kleinbuchstaben. Wenn der Name aus mehreren Worten zusammengesetzt ist, ist CamelCase zu verwenden (panel, panelWidth etc.)

Namen für Konstanten (final-Variablen) dürfen nur aus Großbuchstaben und dem Unterstrich (_) zur Worttrennung bestehen.

5.6 Richtlinien für das Programm-Layout

Neben den Bestimmungen in Abschnitt 5.8 (Konventionen für Kontrollstrukturen) gelten für das Layout des Programms folgende weitere Festlegungen.

Deklarationen von Variablen befinden sich immer am Beginn eines Blocks. Auch wenn diese erst später benutzt werden, ist es aus Gründen der Lesbarkeit sinnvoll, diese an einer festen Stelle zu haben. Weiterhin sollte der Name einer Variable nicht zweimal innerhalb eines Blocks (einmal außerhalb und einmal innerhalb) genutzt werden.

Variablennamen sollten links ausgerichtet sein. Dies dient ebenfalls der Erhöhung der Lesbarkeit von Code:

```
Int       a;
Double    b;
maxValue foo;
```

Eine Ausrichtung empfiehlt sich ebenfalls bei anderen Konstrukten, wie Schleifen oder Verzweigungen. Entwickler sollten dies bei passender Gelegenheit einsetzen.

Wenn auf Attribute zugegriffen wird, so sollen immer die Termini „get“ und „set“ benutzt werden.

5.7 Strukturierungskonventionen

Die Struktur des Quellcodes sollte sich immer an der guten Verständlichkeit für externe Entwickler orientieren. Eine Datei besteht aus einer Klasse oder einem Interface. Private Klassen oder private Interfaces können innerhalb der Datei stehen und befinden sich am Ende der Datei.

Am Beginn der Quellcode-Datei stehen zunächst Kommentare. Diese beinhalten Informationen zur Lizenz. Danach folgen package- und import-Anweisungen.

Schließlich folgt eine Beschreibung der Klasse bzw. des Interface sowie die eigentliche Implementierung.

5.8 Konventionen für Kontrollstrukturen

Kontrollstrukturen dienen der Steuerung des Ablaufs des Programms. In imperativen Sprachen sind das Verzweigungen und Schleifen. Java stellt die Schlüsselworte „if“ und „switch“ für Verzweigungen sowie „for“ und „while“ für Schleifen zur Verfügung. Weiterhin gehören try-catch-Konstrukte ebenfalls in diese Kategorie.

Die if-Anweisung wird in den folgenden Formen angewendet:

```
if (condition) {
statements;
}
if (condition) {
statements;
} else {
statements;
}
if (condition) {
statements;
} else if (condition) {
statements;
} else if (condition) {
statements;
}
```

Es muss immer von geschweiften Klammern ({}) Gebrauch gemacht werden. Denn die Variante ohne Klammern lässt sich schwerer lesen bzw. ist anfälliger für Fehler.

Für switch-Anweisungen ist folgende Form einzuhalten:

```
switch (condition) {
case A:
statements;
case B:
statements;
 break;
case C:
statements;
```

```
 break;
default:
statements;
 break;
}
```

In Fällen, wo kein break enthalten ist, sollte durch einen Kommentar der Grund dokumentiert werden (Fall A).

Die for-Anweisungen werden nach folgendem Muster angewendet:

```
for (initialization; condition; update) {
statements;
}
```

Sollte die Initialisierung mehr als drei Variablen umfassen, so sollte dies in separate Anweisungen vor die for-Schleife ausgelagert werden.

Ein while-Anweisung wird wie folgt angewendet:

```
while (condition) {
statements;
}
```

Schließlich sieht die Vorlage für try-catch-Anweisungen folgendermaßen aus:

```
try {
statements;
} catch (ExceptionClass e) {
statements;
}
```

5.9 Konventionen für Kommentare

Kommentare sind für das Verständnis des Quellcodes und der Überlegungen voriger Entwickler unerlässlich. Daher sollte immer auf eine gute Dokumentation geachtet werden. Die Dokumentation erfolgt in der Regel über Kommentare im Quellcode. Ein guter Kommentar beschreibt nicht, was getan wurde, sondern warum etwas getan wurde.

Es sind zwei Formen von Kommentaren möglich und vorgesehen:

1. Kommentare zur Dokumentation
2. Kommentare bei der Implementierung

Kommentare zur Dokumentation werden in der Form /** … */ angegeben und sind spezifisch für Java. Später können diese Kommentare mittels Javadoc zu HTML-Seiten extrahiert werden.

Kommentare bei der Implementierung beschreiben Informationen, die nicht über den Code zugänglich sind. Der Kommentator sollte hier Informationen hinterlassen, die für das Verständnis des Quellcodes selbst wichtig sind bzw. die Design-Entscheidungen dokumentieren.

D04_QM_Programmierrichtlinien.docxIm Dokument „D02-QM Qualitätskriterien: Aufbau, Messgrößen und Bewertung“ wurde definiert, dass der Anteil undokumentierter Schnittstellen bei 0 % liegen soll. Dabei werden Getter- und Setter-Methoden, leere Konstruktoren und @Override nicht in die Berechnung einbezogen.

Im gleichen Dokument wurde bestimmt, dass der Anteil von Kommentaren im Quelltext über 19 % liegen soll.

Für Kommentare gelten folgende weitere Richtlinien:

- Kommentare sind in englischer Sprache zu verfassen.
- Kommentare sollen in knapper Form Informationen liefern, die nicht direkt aus dem Quelltext ablesbar sind.
- Zur API-Beschreibung sind die zu Verfügung stehenden Javadoc-Tags zu verwenden.

5.10 Paket- und Klassenstruktur

Ein Modul ist eine abgeschlossene Komponente, die über ein genau spezifiziertes Interface ansprechbar ist (vgl. API-Dokumentation).

Die Zerlegung der Gesamtapplikation in Module impliziert die Möglichkeit der getrennten (Weiter-) Entwicklung bzw. des vollständigen Austauschs ganzer Softwareteile (d.h. einzelner oder sogar mehrerer Module), ohne das Anpassungen an anderen Modulen nötig sind.

Zur besseren Organisation fassen wir logisch zusammengehörige Module zu größeren Einheiten wie folgt zusammen:

1. Core (Network-Layer, ISO-24727-Services, eCard-Services)
2. Crypto-Layer
3. GUI-Layer
4. Misc.

Beispielsweise werden die 3 Core-Module „Network-Layer“, „ISO-24727“ und „Card-Services“ zusammengefasst.

Neben den Vorbetrachtungen zu der Modularisierung, gilt es eine Paket- und Klassenstruktur zu entwickeln. Die Paketstruktur kann dem Dokument „D06-QM Architekturkonzept der Open-Source-PersoApp“ entnommen werden.

5.11 Festlegung der maximalen Modulgröße

Ein wichtiges Merkmal einer Softwarekomponente ist ihre Größe. In besonderem Maße gilt dies für „PersoApp“-Software, da sie leicht verteilbar und potentiell auch im mobilen Kontext einsetzbar sein soll. Im Dokument „D02-QM Qualitätskriterien:

Aufbau, Messgrößen und Bewertung“ wurde festgelegt, dass die Größe einer Methode 50 Zeilen nicht überschreiten soll.

6 Interne und externe Anforderungen

Interne Anforderungen an die Entwicklung von Softwarekomponenten im Rahmen des Open-Source-Projekts „PersoApp“ betreffen insbesondere die Einhaltung der Qualitätskriterien, wie sie in „D02-QM Qualitätskriterien: Aufbau, Messgrößen und Bewertung“ festgelegt wurden. In diesem Kontext wurde auch die Einhaltung dieser Programmierrichtlinie gefordert, sowie ein Maßstab zur Bewertung der Konformität vorgegeben.

Externe Anforderungen betreffen im Besonderen die Verwendung und Einbeziehung von externen Softwareprodukten, die im Rahmen dieses Projekts ausschließlich gemäß der vorliegenden Lizenz verwendet werden dürfen.

7 Abläufe

Durch eine sorgfältige Prüfung der hier getroffenen Festlegungen wurde sichergestellt, dass diese Programmierrichtlinie über einen angemessenen Umfang verfügt. Sollte sich dennoch im Projektverlauf die Notwendigkeit für eine Hinzufügung, Anpassung oder Entfernung einer Regelung ergeben, dann ist dies einerseits an alle Projektteilnehmer explizit und verständlich zu kommunizieren, zum anderen gilt es die entsprechende Änderung detailliert festzuhalten und nachvollziehbar zu dokumentieren.

8 Mitgeltende Dokumente

Insbesondere die folgenden Dokumente sind bei der Erstellung und Bewertung von Quelltext im Rahmen des Open-Source-Projekts PersoApp zu berücksichtigen:

- „D02-QM Qualitätskriterien: Aufbau, Messgrößen und Bewertung“
- „D03-QM Entwurfs- und Entwicklungsprozess von sicheren Open-Source-Softwaremodulen der PersoApp“