|  |
| --- |
| Elmer Lukas, Heidt Christina, Steiner Diego, Treichler Delia, Waltenspül Remo  1. Juni 2011 |

|  |
| --- |
| SE2 Projekt MRT |
| Qualitätsmassnahmen |
|  |

****

# Dokumentinformationen

## Änderungsgeschichte

|  |  |  |  |
| --- | --- | --- | --- |
| Datum | Version | Änderung | Autor |
| 04.05.2011 | 1.0 | Erste Version des Dokuments | WR |
| 26.05.2011 | 1.1 | Abschnitt Metriken erstellt | WR |
| 27.05.2011 | 1.2 | Korrekturen | TD |

## Inhaltsverzeichnis

[1 Dokumentinformationen 1](#_Toc294626179)

[1.1 Änderungsgeschichte 1](#_Toc294626180)

[1.2 Inhaltsverzeichnis 1](#_Toc294626181)

[1.3 Abbildungsverzeichnis 2](#_Toc294626182)

[2 Einführung 3](#_Toc294626183)

[2.1 Zweck 3](#_Toc294626184)

[2.2 Gültigkeitsbereich 3](#_Toc294626185)

[2.3 Definitionen und Abkürzungen 3](#_Toc294626186)

[2.4 Referenzen 3](#_Toc294626187)

[3 Tests 4](#_Toc294626188)

[4 Codestatistiken 5](#_Toc294626189)

[4.1 Android 5](#_Toc294626190)

[4.2 Rails 6](#_Toc294626191)

[5 Codemetriken 8](#_Toc294626192)

[5.1 Server 8](#_Toc294626193)

[5.2 Client 8](#_Toc294626194)

[5.2.1 State of Flow Eclipse Metrics 8](#_Toc294626195)

[5.2.2 Structure 101 13](#_Toc294626196)

[5.2.3 Schlussfolgerung 15](#_Toc294626197)

[6 Codereview 16](#_Toc294626198)

[6.1 Ablauf 16](#_Toc294626199)

[6.2 Fokus bei Review 16](#_Toc294626200)

[6.3 Eingesetzte Tools 16](#_Toc294626201)

[6.3.1 Findbugs 16](#_Toc294626202)

[6.3.2 Checkstyle 16](#_Toc294626203)

[6.3.3 State Of Flow Eclipse Metrics Plugin 16](#_Toc294626204)

[6.3.4 Structure 101 16](#_Toc294626205)

[6.4 Reviews Android 18](#_Toc294626206)

[6.4.1 Review 1 (04.05.2011) 18](#_Toc294626207)

[6.4.2 Review 2 (24.05.2011) 18](#_Toc294626208)

[6.5 Reviews Rails 20](#_Toc294626209)

[6.5.1 Review 1 (08.05.2011) 20](#_Toc294626210)

[7 Übereinstimmung mit Planung 21](#_Toc294626211)

[7.1 Design 21](#_Toc294626212)

[7.2 Coderichtlinien 21](#_Toc294626213)

[7.2.1 Anpassungen 21](#_Toc294626214)

## Abbildungsverzeichnis

[Abbildung 1 - Übersicht Codestatistik Android 5](#_Toc294628522)

[Abbildung 2 - Codestatistik Android 5](#_Toc294628523)

[Abbildung 3 - Codestatistik Rails 6](#_Toc294628524)

[Abbildung 4 - Übersicht über die 40 benutzten Libraries 7](#_Toc294628525)

[Abbildung 5 - Übersicht Cyclomatic Complexity 9](#_Toc294628526)

[Abbildung 6 - Übersicht Lines of Code 10](#_Toc294628527)

[Abbildung 7 - Übersicht Weighted Methods per Class 11](#_Toc294628528)

[Abbildung 8 - Codebeispiel Lifecycle Methoden 11](#_Toc294628529)

[Abbildung 9 - Übersicht Efferent Couplings 12](#_Toc294628530)

[Abbildung 10 - Ausschnitt Importliste Klasse TimeEntryActivity 12](#_Toc294628531)

[Abbildung 11 - Abhängigkeiten zwischen Packages 13](#_Toc294628532)

[Abbildung 12 - Abhängigkeiten in Package Services 13](#_Toc294628533)

[Abbildung 13 - Abhängigkeiten in Package Activity 14](#_Toc294628534)

[Abbildung 14 - Abhängigkeiten in Package Models 15](#_Toc294628535)

[Abbildung 15 - Abhängigkeiten in Package Network 15](#_Toc294628536)

# Einführung

## Zweck

Der Zweck dieses Dokuments ist die Aufzeigung von Codestatistiken und -metriken, die Dokumentation der Reviews und Festhaltung der Änderungen gegenüber der Planung bezüglich Design und Coderichtlinien für das Projekt MRT (Mobile Reporting Tool).

## Gültigkeitsbereich

Dieses Dokument ist während der gesamten Projektdauer gültig (21.02 bis 03.06.2011).

## Definitionen und Abkürzungen

Die Definitionen und Abkürzungen befinden sich in der ausgelagerten Datei doc/01\_Projektplan/glossar.docx.

## Referenzen

* doc/11\_Qualitaetsmassnahmen/metrics\_client/\*
* doc/11\_Qualitaetsmassnahmen/metrics\_client\_without\_test/\*
* doc/11\_Qualitaetsmassnahmen/structure101/\*
* doc/06\_Tests/ms6/test\_dokumentation\_ms6.docx
* doc/05\_Design/software\_architektur.docx

# Tests

Die Tests wurden ausgelagert in die Datei test\_dokumentation\_ms6.docx.

# Codestatistiken

## Client

Die Codestatistiken des Clients können nicht mit jenen des Servers verglichen werden, da in Ruby für die Programmierung der gleichen Funktionalität oftmals weniger Code benötigt wird.

Die generierten Codezeilen sind alle in der Klasse R.java enthalten, welche die Programmierschnittstelle für den View-Layer von Android darstellt.

|  |  |  |
| --- | --- | --- |
| Posten | Anzahl | Davon Tests |
| Anzahl Packages | 10 | 2 |
| Anzahl eigene Klassen | 59 | 25 |
| LOC (Lines Of Code) | 3305 | 1422 |
| NI (Number of bytecode Instructions) | 7685 | 3306 |
| Generierte Codezeilen | 66 | 0 |

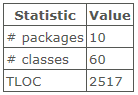


Abbildung 1 - Übersicht Codestatistik Android

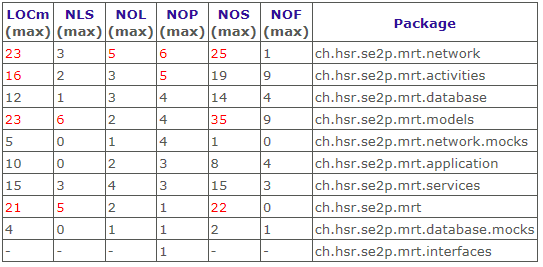


Abbildung 2 - Codestatistik Android

Bildlegende zu Abbildung 2 - Codestatistik Android:

LOCm Lines of Code in Method

NLS Number of Locals in Scope

NOL Number of Levels

NOP Number of Parameters

NOS Number of Statements

NOF Number of Fields

## Server

Die Codestatistiken des Servers können nicht mit jenen des Clients verglichen werden, da in Ruby für die Programmierung der gleichen Funktionalität oftmals weniger Code benötigt wird.

Ein Teil der Programmierarbeit des Servers ist auch in die Erstellung von HTML-, CSS- und Übersetzungs-Files geflossen.

|  |  |
| --- | --- |
| Posten | Anzahl |
| Anzahl eigene Klassen | 24 |
| Geschriebene Codezeilen |  |
| Applikation | 1290 |
| Tests | 1151 |
| Views / XHTML | 1111 |
| CSS | 464 |
| Übersetzungen | 556 |
| Total | 4572 |
| Generierte Codezeilen | ~100 |
| Code / Test Ratio | 1.1/1 |

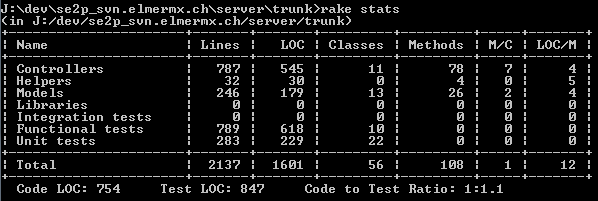


Abbildung 3 - Codestatistik Rails

Untenstehend eine Übersicht über die 41 benutzten Libraries, unterteilt nach Kategorie:

|  |  |  |
| --- | --- | --- |
| **Deployment** | | |
| \* capistrano (2.6.0) |  |  |
| **Testing** | | |
| \* rcov (0.9.9)    \* ruby-prof (0.10.7)    \* railroady (1.0.2)    \* sqlite3 (1.3.3) |  |  |
| **Server monitoring / debugging** | | |
| \* exception\_notification (2.4.0) |  |  |
| **Authentication** |  |  |
| \* devise (1.3.4)    \* warden (1.0.4) |  |  |
| **Database** |  |  |
| \* mysql (2.8.1) |  |  |
| **Rails** |  |  |
| \* abstract (1.0.0)    \* actionmailer (3.0.5)    \* actionpack (3.0.5)    \* activemodel (3.0.5)    \* activerecord (3.0.5)    \* activeresource (3.0.5)    \* activesupport (3.0.5)    \* arel (2.0.10)    \* bcrypt-ruby (2.1.4)    \* dynamic\_form (1.1.4)    \* erubis (2.6.6)    \* highline (1.6.2) | \* i18n (0.6.0)    \* mail (2.2.19)    \* mime-types (1.16)    \* net-scp (1.0.4)  \* net-sftp (2.0.5)    \* net-ssh (2.1.4)    \* net-ssh-gateway (1.1.0)    \* orm\_adapter (0.0.5)    \* polyglot (0.3.1)    \* rack (1.2.2)    \* rack-mount (0.6.14)    \* rack-test (0.5.7) | \* rails (3.0.5)    \* railties (3.0.5)    \* rake (0.8.7)  \* thor (0.14.6)    \* treetop (1.4.9)    \* tzinfo (0.3.27)    \* bundler (1.0.13)    \* builder (2.1.2) |

# Codemetriken

Für die Metrikanalyse des implementierten Codes wurden unter Android die Anwendungen State of Flow Eclipse Metrics sowie Structure 101 verwendet.

Die nachfolgend beschriebenen Metrik-Indikatoren wiederspiegeln nur die interessantesten Werte bzw. Auffälligkeiten. Um eine ausführliche Metrik-Analyse zu erhalten, wird auf die unter dem Kapitel 2.4 Referenzen aufgeführten Ordner verwiesen.

## Server

Im Team wurde beschlossen auf die Metrik-Analyse des Servers zu verzichten. Dieser Entscheid wurde aufgrund der weniger hohen Komplexität im Vergleich zum Client sowie der mangelnden Zeit gefällt.

## Client

Da die Testklassen für die Sicherstellung von diversen Funktionen verantwortlich sind, sind die Kohäsionswerte zum Teil geringer als es gewünscht wäre. In der Regel wurde für jede zu testende Klasse eine Testklasse entworfen. Eine weitere Aufteilung auf mehrere Klassen wäre nicht sinnvoll, da die Testklassen nur für den Test erzeugt werden und keinen weiteren Nutzen haben.

Aus diesen Gründen wird in den nachfolgenden Unterkapiteln eine Analyse des Codes exklusive der durchgeführten Tests vorgenommen.

### State of Flow Eclipse Metrics

Das Eclipse-Plugin State of Flow Eclipse Metrics wird verwendet, um diverse Code Qualitätsindikatoren wie zum Beispiel „McCabe’s Cyclomatic Complexity“, „Efferent Coupling“, „Feature Envy“, „Lines of Code in Method“ etc. aus dem Code zu berechnen. Diese Indikatoren werden zudem laufend während der Entwicklung neu ausgewertet und direkt in der IDE als Marker angezeigt.

#### Cyclomatic Complexity

Wie aus der nachfolgenden Abbildung 5 - Übersicht Cyclomatic Complexity ersichtlich, ist die zyklomatische Komplexität relativ gering. Sämtliche Klassen befinden sich innerhalb des gewünschten Bereichs von eins bis vier.

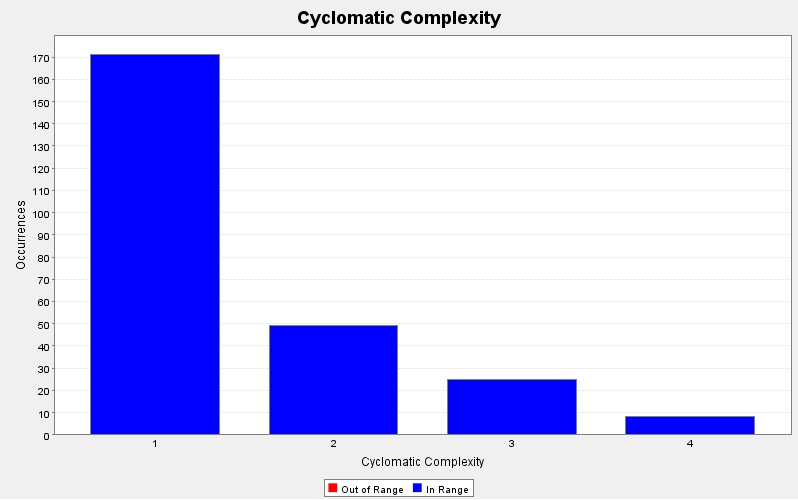


Abbildung 5 - Übersicht Cyclomatic Complexity

#### Lines of Statements

Der Grenzwert für die Anzahl Anweisungen in einer Methode wurde auf 20 festgelegt. Alle Methoden liegen innerhalb des erwarteten Wertes (siehe Abbildung 6 - Übersicht Lines of Code).

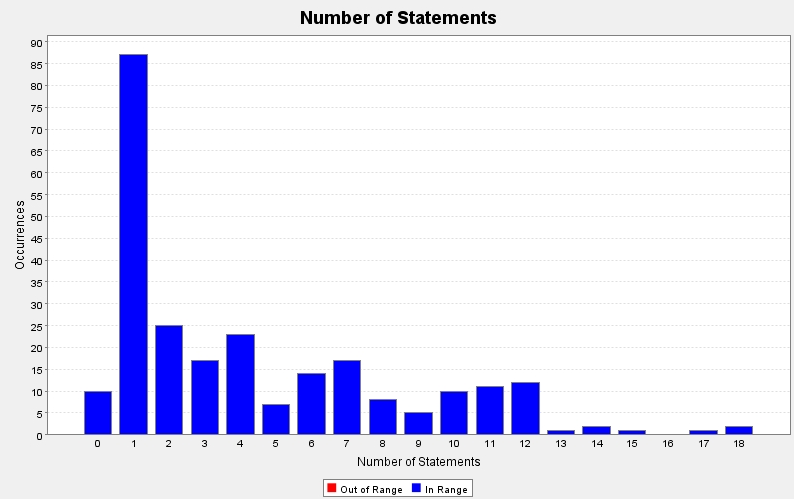


Abbildung 6 - Übersicht Lines of Code

#### Weighted Methods per Class

Der Metrik-Indikator „Weighted Methods per Class“ steht für die kumulierte Komplexität sämtlicher Methoden einer Klasse. Das Eclipse Metric Plugin verwendet für die Komplexitätsanalyse den MC Cabe’s Algorithmus. Je höher der Wert liegt, desto höher ist die Komplexität einer kompletten Klasse. Der gewünschte Bereich für diese Kenngrösse liegt zwischen 0 und 40.

Wie die nachstehenden Abbildung 7 - Übersicht Weighted Methods per Class zeigt, existieren zwei Klassen, welche den definierten Grenzwert mit dem Wert 41 knapp überschreiten.

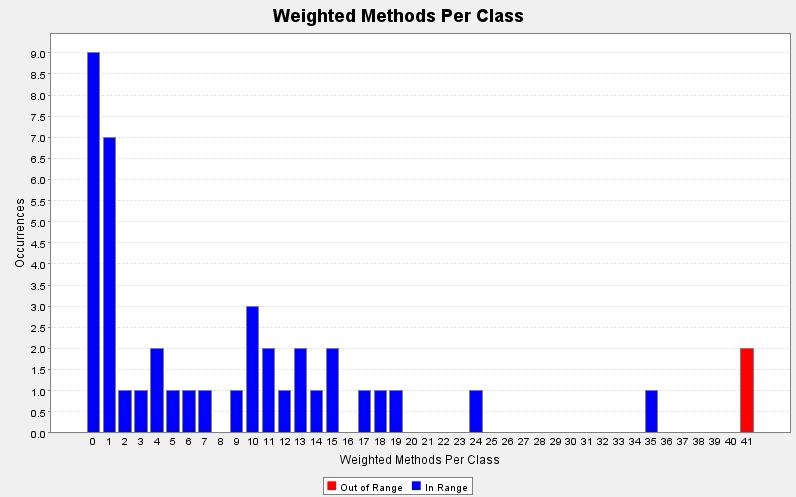
****

Abbildung 7 - Übersicht Weighted Methods per Class

**Erklärung hohe Komplexität**

Bei den Klassen mit überschrittener Komplexität handelt es sich um die Klasse TimeEntryActivity sowie die Klasse Customer. Der Schwellwert wird nur um eins übertreten.

Bei der Klasse TimeEntryActivity sind der Grund dafür in erster Linie die vielen kurzen Methoden, welche diese Klasse enthält, da jede Methode die Komplexität der Klasse um eins inkrementiert. Es wurde darauf verzichtet, diese Klasse weiter auf neue Klassen aufzuteilen, da die Kohäsionswerte im grünen Bereich liegen sowie viele Methoden, wie zum Beispiel die Lifecycle Methoden, fast unumgänglich sind (siehe Abbildung 8 - Codebeispiel Lifecycle Methoden).

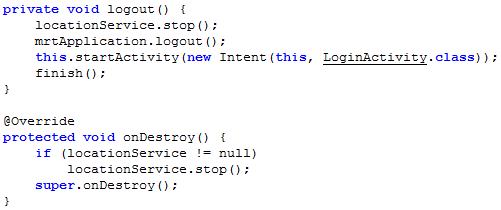


Abbildung 8 - Codebeispiel Lifecycle Methoden

Im Falle der zweiten Klasse Customer ist der Hauptgrund für die hohe Komplexität der Einsatz von Settern bzw. Gettern für diverse Attribute.

#### Efferent Coupling

Der Efferent Coupling Wert beschreibt die Anzahl verwendeter Typen in einer Klasse. Dies kann ein Indikator für die Kopplung einer Klasse darstellen.

Die nachfolgende Abbildung 9 - Übersicht Efferent Couplings verdeutlicht, dass, abgesehen von einer Klasse, sämtliche Klassen im gewünschten Bereich von 1 bis 25 liegen.

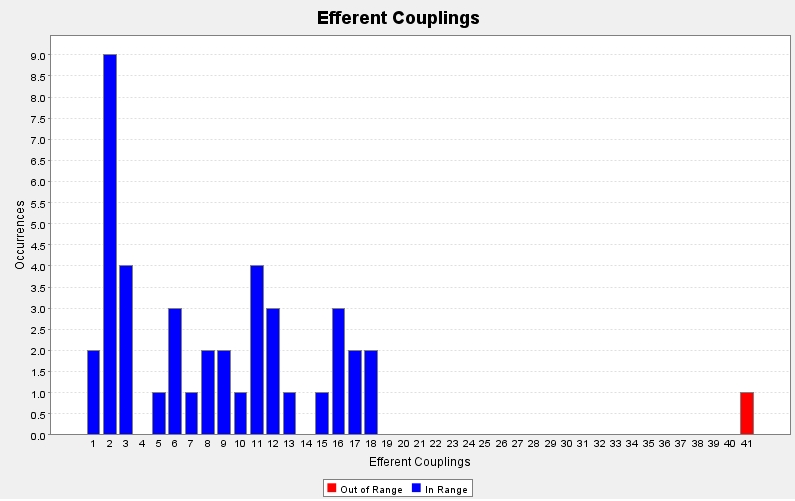


Abbildung 9 - Übersicht Efferent Couplings

**Erklärung Hohe Kopplung Klasse TimeEntryActivity**

Die Klasse TimeEntryActivity ist für die Visualisierung der Hauptanzeige (User Interface) verantwortlich. Dementsprechend enthält diese Klasse eine grosse Anzahl referenzierter GUI Typen. Diese sind in erster Linie der Grund für die hohe Kopplung (siehe Abbildung 10 - Ausschnitt Importliste Klasse TimeEntryActivity). Die Anbindung an diese Komponenten ist jedoch nicht weiter tragisch, da es sich bei den GUI Typen um Klassen handelt, die im Android Framework integriert sind und stabil laufen.

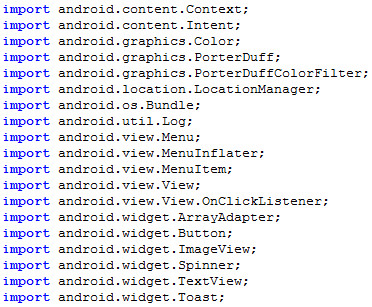


Abbildung 10 - Ausschnitt Importliste Klasse TimeEntryActivity

### Structure 101

#### Übersicht

Bei der Entwicklung der Anwendung MRT wurde darauf geachtet, dass keine zirkuläre Abhängigkeiten zwischen den Packages bestehen. Wie aus der nachstehenden Abbildung 11 - Abhängigkeiten zwischen Packages ersichtlich wird, bestehen keine zirkulären Abhängigkeiten auf Package-Ebene.

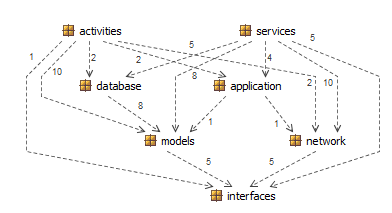


Abbildung 11 - Abhängigkeiten zwischen Packages

#### Abhängigkeiten innerhalb Package Services

Das Package Services enthält die Klassen zur Synchronisation von Stundeneinträgen, Stundeneintragstypen sowie Kunden. Wie aus der Abbildung 12 - Abhängigkeiten innerhalb Package Services hervorgeht, existiert in diesem Paket eine zirkuläre Abhängigkeit zwischen den beiden Klassen SynchronizationServiceTask und SychnronizationService. Ansonsten verhalten sich die Abhängigkeiten hierarchisch, wobei die Klasse SynchronizationService die Wurzel darstellt.

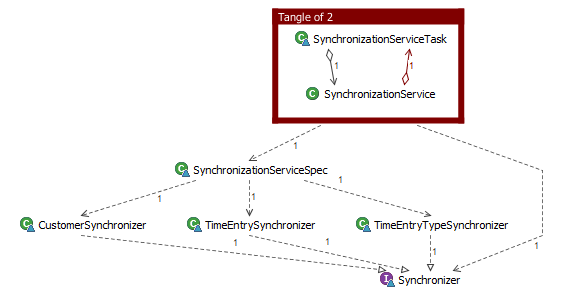


Abbildung 12 - Abhängigkeiten innerhalb Package Services

**Erklärung zirkuläre Abhängigkeit**

Die Klasse SynchronizationService enthält Lifecycle Methoden wie zum Beispiel onCreate(), onStart() etc., welche beim Starten des Dienstes aufgerufen werden (für Codebeispiel siehe Abbildung 8 - Codebeispiel Lifecycle Methoden). Die Klasse instanziiert ein Objekt der SynchronisationServiceTask, damit sie möglichst kohäsiv bleibt. Das neu erstellte Objekt benötigt jedoch eine Referenz auf den Service selbst, da der Kontext bzw. die Datenbankhelper an den Service gebunden sind.

#### Abhängigkeiten innerhalb Package Activities

Das Package Activities enthält Klassen, welche für die Darstellung der Benutzeroberfläche zuständig sind. Anhand der nachstehenden Abbildung 13 - Abhängigkeiten innerhalb Package Activity wird eine zirkuläre Abhängigkeit sichtbar, welche untenstehend begründet wird.

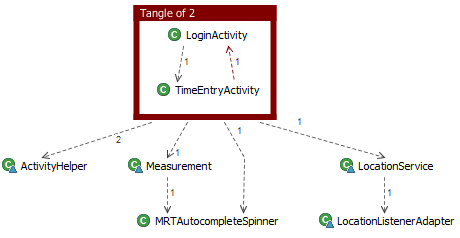


Abbildung 13 - Abhängigkeiten innerhalb Package Activity

**Erklärung zirkuläre Abhängigkeit**

Diese zirkuläre Abhängigkeit ist durch das Android Frameworks vorgegeben, da beim Aufruf einer neuen Activity der Context einer Applikation übergeben werden muss. Deshalb wird die Referenz auf das eigene Objekt (this) übergeben.

#### Abhängigkeiten innerhalb Package Models

Wie in der nachfolgenden Abbildung 14 - Abhängigkeiten innerhalb Package Models dargestellt wird, kommen keine zirkulären Abhängigkeiten im Package Models vor.

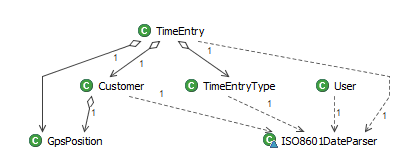


Abbildung 14 - Abhängigkeiten innerhalb Package Models

#### Abhängigkeiten innerhalb Package Network

Im Package Network sind keine zirkulären Abhängigkeiten zwischen den Klassen vorhanden (siehe Abbildung 15 - Abhängigkeiten innerhalb Package Network). Es herrscht eine strenge Top-Down-Hierarchie.

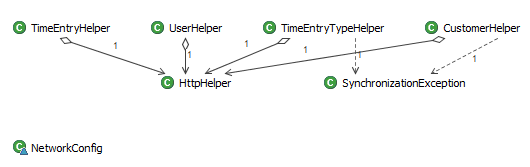


Abbildung 15 - Abhängigkeiten innerhalb Package Network

### Schlussfolgerung

Wie aus den vorangehenden Unterkapiteln ersichtlich ist, sind, bis auf wenige Ausnahmen, keine zirkulären Abhängigkeiten in der Anwendung präsent. Zudem befinden sich die Codemetrik-Indikatoren, abgesehen von zwei Klassen, in einem erwarteten Bereich.

Aufgrund dieser Messungen werden keine weiteren Änderungen an der Architektur bzw. dem Code durchgeführt.

# Codereview

## Ablauf

Ein Review wird mindestens zu zweit durgeführt. Ein Teilnehmer des Reviews ist die Person, welche den ausgewählten Code geschrieben hat.

Der Review-Vorgang besteht aus dem Durchschauen des Codes und der Anwendung der unter Kapitel 3 „Eingesetzte Anwendungen, Plug-Ins“ erwähnten Software-Produkten.

Während des Reviews wird eine Liste geführt, in der alle gefundenen Ungereimtheiten und Fehler dokumentiert werden.

Nach dem Review üben die in der Liste zugewiesenen Personen die Fehlerbehebung oder das Refactoring für die bestimmten Codeabschnitte aus.

## Fokus bei Review

* Code auf Verständnis überprüfen
* Code Smells
* Fehler finden
* Namenswahl (aussagekräftige Namen für Methoden, Klassen, Packages)
* Korrekte Abbildung von Architekturentscheidungen

## Eingesetzte Tools

### Findbugs

Findbugs ist eine Open-Source Code-Analyse-Software, sie untersucht Java-Programme nach Fehlermustern. Es existiert ein Eclipse Plug-in für eine interaktive Nutzung.

Die Suche nach Fehlern geschieht ausschliesslich durch das Prüfen des Bytecodes, das Programm muss dazu nicht ausgeführt werden.

### Checkstyle

Checkstyle überprüft die Einhaltung von Coding Conventions bei der Erstellung von Java-Code. Convention-Verletzungen werden als Warnungen, nach Kategorie sortiert, angezeigt. Das Programm erlaubt einem, einen einheitlichen Coding Standard über alle Implementierungsarbeiten hinweg zu erreichen und so die Softwarequalität zu erhöhen.

### State Of Flow Eclipse Metrics Plugin

Das State Of Flow Eclipse Metrics Plugin berechnet verschiedenste Metriken bei der Überprüfung des Codes. Es gibt Warnungen für jede Metrik aus, bei der der Schwellwert überschritten ist.

* Das Exportieren eines HTML-Reports ist möglich, er bietet eine gute Übersicht über alle Warnungen pro Metrik des überprüften Codes.

### Structure 101

Structure 101 bietet eine Fülle von Funktion und Möglichkeiten zur Analyse des Codes sowie den Interaktionen bzw. Abhängigkeiten zwischen Packages und Klassen. Diese Anwendung wird insbesondere für die Darstellung der Abhängigkeiten zwischen den einzelnen Packages verwendet. Falls zirkuläre Abhängigkeiten zwischen Packages bestehen, werden diese als „Tangles“ identifiziert.

## Reviews Android

### Review 1 (04.05.2011)

Review der Klasse LoginActivity

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| ID | Beschreibung | Schweregrad | Datum & Kürzel wenn behoben | Req. Ref. | Bemerkungen |
| 1 | Methode processLogin() ist zu umfangreich (smell Long Method) | Mittel | 04.05.2011 WR | - | Refactoring ExtractMethod |
| 2 | Ähnlicher Code wird mehrmals aufgerufen (smell Duplicated Code) | Mittel | 04.05.2011 WR | Ja | Refactoring ExtractMethod |
| 3 | Attributnamen zu wenig aussagekräftig | Leicht | 04.05.2011 WR | - | Refactoring RenameField |
| 4 | Name für Methode „checkPref“ zu wenig aussagekräftig | Leicht | 04.05.2011 WR | - | Refactoring RenameMethod |
| 5 | Zugriffsmodifikator für Methode switchToTimeEntryActivity() ist auf protected gesetzt | Mittel | 04.05.2011 WR | Ja | - |

#### Referenz ID 2

Die Objekte, welche die GUI Schaltflächen repräsentieren, werden bei jedem Zugriff neu geladen, anstatt diese direkt als Attribute der Klasse zu speichern.

#### Referenz ID 5

Die Methode switchToTimeEntryActivity() wird nur von der eigenen Klasse verwendet, dennoch steht der Zugriffsmodifikator auf protected. Dies ist unter Umständen ein Sicherheitsdefizit, welches auch unter dem Namen „Inappropriate Intimacy“ als Code Smell bekannt ist.

#### Verdikt

Das vorliegende Review wurde vom Team akzeptiert.

### Review 2 (24.05.2011)

Review der Klasse TimeEntryAcivity

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| ID | Beschreibung | Schweregrad | Datum & Kürzel wenn behoben | Req. Ref. | Bemerkungen |
| 1 | Klasse zu gross, zusätzliche Klassen für Zeitmessung, LocationService und Spinner erstellen | Hoch | 24.05.2011 EL | Ja | Diverse Refactorings |
| 2 | Zwei verschiedene Spinners vorhanden  -> Attributnamen zu wenig aussagekräftig | Leicht | 24.05.2011 EL | - | Refactoring RenameField |
| 3 | Methodennamen update() zu wenig aussagekräftig | Leicht | 24.05.2011 EL | - | Refactoring RenameMethod |
| 4 | Zurücksetzen der Daten nach dem Stoppen der Zeitmessung in eine eigene Methode extrahieren | Leicht | 24.05.2011 EL | - | Refactoring ExtractMethod |

#### Referenz ID 1

Die Klasse TimeEntryActivity hat zu viele Verantwortlichkeiten. Die Zeitmessung, der Location Service und der Spinner mit der Autocompletion-Funktion sollen je in eine eigene Klasse ausgelagert werden.

#### Verdikt

Das vorliegende Review wurde vom Team akzeptiert.

## Reviews Rails

### Review 1 (08.05.2011)

Review sämtlicher Models

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| ID | Beschreibung | Schweregrad | Datum & Kürzel wenn behoben | Req. Ref. | Bemerkungen |
| 1 | Kompliziertes Konstrukt anstelle von Polymorphismus verwendet (Model User => for\_select) | Mittel | 08.05.2011 EL | - |  |
| 2 | Auflösung Model in eine schriftliche Darstellung wird in View gemacht (Duplicated Code) sämtliche Models | Mittel | 08.05.2011 EL | Ja | Extract Method |
| 3 | Es werden Stundeneintragstypen angezeigt, welche nicht mehr gültig sind  (Model TimeEntryType) | Schwer | 09.05.2011 EL | Ja | scope :active |
| 4 | Beim Kunden wird nicht geprüft, ob Nachname angegeben wurde  (Model Customer) | Leicht | 08.05.2011 EL | - | validates :last\_name |
| 5 | Beziehung zwischen Model AppliedMaterial und Material falsch gesetzt. | Mittel | 08.05.2011 EL | - | Model AppliedMaterial has\_many anstatt belongs\_to |

#### Referenz ID 2

Statt einer Methode auf Modelebene (to\_s) um das Model als String darzustellen, wird bei jedem Gebrauch der String mit den passenden Attributen erstellt. Nach Konvention soll eine Methode erzeugt werden, die das Model als String repräsentieren kann (to\_s).

#### Referenz ID 3

Es ist möglich einen Stundeneintragstyp als ungültig zu definieren, jedoch sollten solche Stundeneintragstypen anschliessend bei der Auswahl nicht mehr zur Verfügung stehen. Um dies zu realisieren könnte die Methode scope verwendet werden, welche nur bestimmte Datensätze anzeigt.

#### Verdikt

Das vorliegende Review wurde vom Team akzeptiert.

## Diverse Reviews

Während der gesamten Projektzeit wurden stetig nichtformale Reviews durch die Projektmitglieder durchgeführt. Diese wurden in Form von Redmine-Tickets oder in den Sitzungsprotokollen festgehalten.

# Übereinstimmung mit Planung

## Design

Die Implementation stimmt mit dem Design überein, so wie es im Dokument software\_architektur.docx festgehalten ist.

## Coderichtlinien

Die Coderichtlinien des Projektes wurden eingehalten.

### Anpassungen

Während der Construction-Phase ist die Erfahrung gemacht worden, dass die zuvor festgelegte Spaltenbreite für die von uns eingesetzten Bildschirme zu wenig breit ist. Bei der automatischen Formatierung wurden so Zeilen umgebrochen, obwohl die ganze Zeile ohne weiteres noch auf derselben Linie Platz gehabt hätte. Die Coderichtlinien wurden dem entsprechend angepasst.