TECHNISCHE UNIVERSITÄT DRESDEN

Software Technology Group

Fakultät Informatik Institut für Software- und Multimediatechnik, Lehrstuhl für Softwaretechnologie

Großer Beleg

DURCHFÜHRUNG EINER ENTWICKLERSTUDIE ZUM ERMITTELN VON QUALITY SMELLS UND DEREN BESEITIGUNG AUF ANDROID-SYSTEMEN

bearbeitet von
Martin Brylski
geboren am 17.02.1986 in Dresden

Betreuer:
Dipl.-Inf. Jan Reimann

Hochschullehrer:
Prof. Dr. rer. nat. habil. Uwe Aßmann

Eingereicht am 27. März 2014

Inhaltsverzeichnis

1 Einführung

With little power comes great responsibility.

(Hervé Guihot)

1.1 Motivation

In den letzten Jahren ist der Anwendungsbereich und die Zahl mobiler Anwendungen (MA) stetig gestiegen. Man kann mit ihnen spielen, navigieren, E-Mails schreiben, Bücher lesen, videotelefonieren und die Fitness überwachen. Sie messen Position, Lage, Beschleunigung und Puls, nehmen Fotos auf und Sprachbefehle entgegen. Die Anzahl der Smartphone-Nutzer allein in Deutschland ist in den letzten vier Jahren um das fünffache, auf 35,7 Millionen, gestiegen (siehe Abbildung 1.1). Aufgrund unterschiedlicher Hardwarebeschränkungen, wie Prozessor- und Akkuleistung, sowie Größe des Arbeitsspeichers und des Bildschirms haben Benutzer hohe Anforderung an MA, wie Rechen-, Energie- und Speichereffizienz oder Bedienbarkeit. Laut [Dra12] sind neben dem "Nichterfüllen von Erwartungen" (58,6%), das "Beeinflussen der Geschwindigkeit vom Handy" (42,5%) und "Bei Nutzung hoher Leistungsverbrauch" (25,9%) die häufigsten Gründe für das Deinstallieren von MA. Nach einer Studie von [Wil+13] hat jede sechste Android-Anwendung Probleme mit dem Energieverbrauch. Damit sind die Entwickler in der Pflicht die Software so zu gestalten, dass diese Anforderungen in für den Benutzer ausreichendem Maße erfüllt werden.

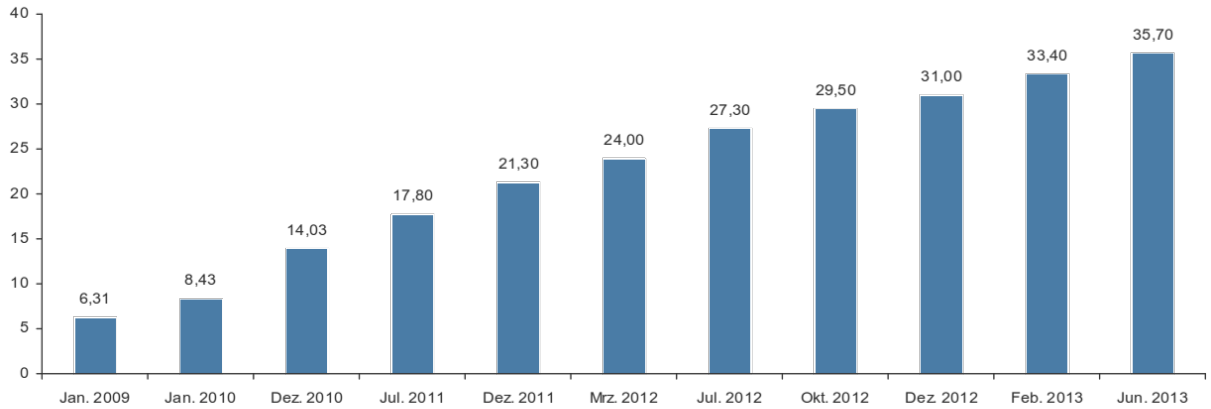


Abbildung 1.1: *Anzahl der Smartphone-Nutzer in Deutschland in den Jahren 2009 bis 2013 (in Millionen), Quelle: [Mob13]*

Während der Anforderungsanalyse in einem Softwareentwicklungsprozess werden alle Bedürfnisse des Kunden an die Software erarbeitet, spezifiziert und analysiert [Bal09]. Das sind einerseits *funktionelle Anforderungen*, welche beschreiben, was die Anwendung leisten soll und andererseits *nichtfunktionale Anforderungen*, die beschreiben, welche Qualitätseigenschaften die Software erfüllen soll. Diese Anforderungen werden dann in der Implementierungsphase realisiert. In einer Optimierungsphase wird die Anwendung so verändert, dass diese Softwarequalitäten (SQ) besser erfüllt werden. Fowler u. a. [Fow+12] prägten dabei den Begriff *Refactoring*, als Umstrukturierung von Code, ohne dabei die Funktionalität zu verändern und den Begriff *Bad Smell*, als Indikator für eine notwendige Änderung. Das Konzept der *Quality Smells* von Reimann und Aßmann [RA13] verbindet *Bad Smells* und *Refactorings* mit ihrer Auswirkung auf SQ und legt damit den Fokus auf MA. Denn bei Festrechnern, bei denen leicht Hardwareelemente aktualisiert und erweitert werden können, spielt Effizienz, zum Beispiel beim Stromverbrauch, eine eher untergeordnete Rolle. Bei mobilen Geräten jedoch, die über beschränkte und fest verbaute Hardwareressourcen verfügen, ist sie wichtiger. Akzeptanztests zeigen Probleme bei der Umsetzung von Qualitätsanforderungen auf. Bisher haben Entwickler, durch Best Practices, Forendiskussionen und Bug-Tracker, eine eher indirekte Vorstellung, welche *Bad Smells* SQ negativ beeinflussen. Das ist aber hinderlich für das Optimieren von Software. Außerdem fehlen Werkzeuge für Entwickler, um qualitätsorientierte Softwareentwicklung schon frühestmöglich umzusetzen und die komplexe Optimierungsphase zu automatisieren. Denn *Quality Smells* betreffen meist nicht nur ein Codeartefakt, sondern das ganze Softwaresystem [HB10]. Als Voraussetzung, um ein Werkzeug zur Verfügung zu stellen, muss ein Katalog von *Quality Smells* erarbeitet werden, in denen dokumentiert wird, wie sie identifiziert werden, welche Qualitäten beeinflusst werden und welche *Refactorings* sie beseitigen. Allerdings ist es, durch die Verteilung der Erfahrungswerte von Softwareentwicklern über das gesamte Internet schwer, alle zu sammeln und anzuwenden.

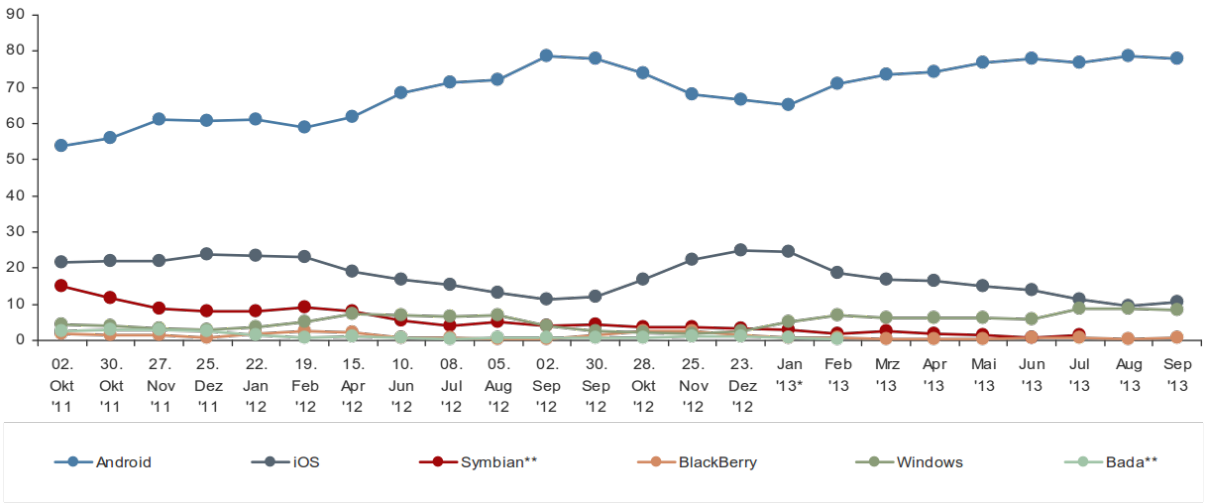


Abbildung 1.2: *Anteile der Smartphone-Betriebssysteme am Absatz in Deutschland bis September 2013, Quelle: [Com13]*

Diese Arbeit stellt nun einen Katalog mit 30 *Quality Smells* vor. Aufgrund der weiten Verbreitung von Android (siehe Abbildung 1.2) und der großen Community wurden dafür

nur Internetplattformen dieses Systems durchsucht. Ausgewählte *Bad Smells* wurden im Eclipse[1]-Werkzeug Refactory[2] umgesetzt.

1.2 Übersicht

Kapitel 2 gibt eine Einführung in grundlegende Themen, wie der Android Plattform, Qualitäten in der Softwareentwicklung und des Konzeptes der *Quality Smells*.

Kapitel 3 beschreibt Art der Studie und warum diese gewählt wurde. Es wird geschildert, warum die Webplattformen ausgewählt und wie sie durchsucht und analysiert wurden. Anschließend wird das Gesamtergebnis ausgewertet.

Kapitel 4 präsentiert detailliert die 30 *Quality Smells* als Ergebnis der Studie. Dabei werden Codebeispiele und weitere Referenzen genannt.

Kapitel 5 beschreibt wie *Quality Smells* in einem Werkzeug umgesetzt werden können und gibt ein beispielhafte Implementierung.

Kapitel 6 fasst die Arbeit zusammen und verweist auf offene Fragen.

[1] http://www.eclipse.org
[2] http://www.modelrefactoring.org

2 Grundlagen

2.1 Android

Android[1] ist eine freie, quelloffene Software und wird von der *Open Handset Alliance*, deren Hauptmitglied Google ist, entwickelt. Darüber hinaus gehören Netzbetreiber, sowie Firmen der Halbleiterindustrie, Softwareentwicklung, des Marketings und Geräteherstellers dazu. Android ist einerseits ein Betriebssystem, andererseits eine Softwareplattform zur Entwicklung MA für Smartphones, Tablets, Navigationsgeräte, Multimediageräte und Kameras.

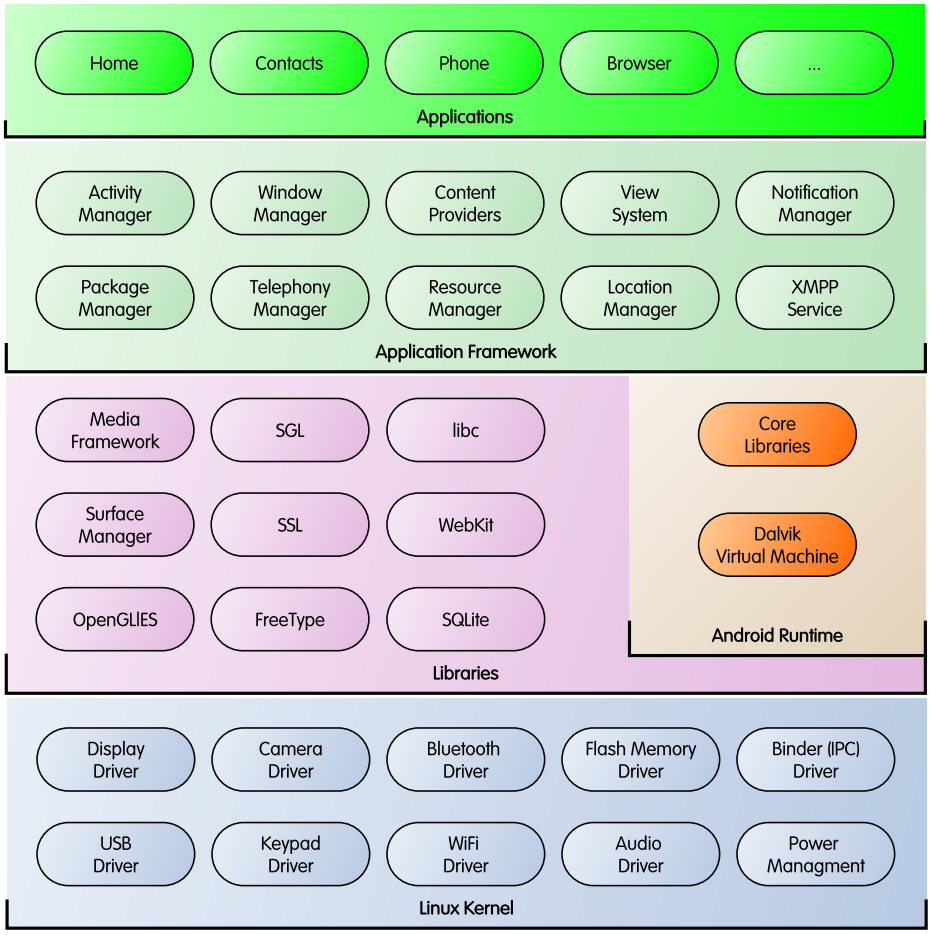


Abbildung 2.1: *Architektur des Android Systems, [Wik13a] nachempfunden*

[1] http://www.android.com

In Abbildung 2.1 ist die Architektur des Android Systems zu sehen. Es basiert auf einem Linux Kernel, der für die Hardwareinteraktion und Prozesssteuerung zuständig ist. Die nächste Schicht besteht aus nativen Bibliotheken[2], zum Beispiel für die 2D&3D-Grafikausgabe, den Datenbankzugriff, die Audio- und Videokodierung, sowie Webseitendarstellung. Die Android Laufzeitumgebung[3] besteht aus der *Dalvik Virtual Machine*, eine von Google entwickelte virtuelle Maschine für Java, mit besonderem Schwerpunkt auf Ressourcenschonung und den *Core Java Libraries*, welche speziell für die Entwicklung von Android Anwendungen entworfen worden. Darauf baut das *Application Framework* auf. Es enthält alle Bestandteile mit denen MA mit dem System interagieren können. Die am häufigsten genutzten Anwendungskomponenten sind:

Activity Entspricht einem einzelnen Fenster des UI einer Anwendung. Zum Beispiel kann ein E-Mail Programm drei *Activities* besitzen, zum Auflisten, Anschauen und Schreiben von Nachrichten. Der *Activity Manager* verwaltet den Lebenszyklus. Das Aussehen des UI wird über eine XML-Datei festgelegt.

Service Ein Service läuft im Hintergrund, hat keine Benutzeroberfläche und verrichtet lang andauernde Operationen, wie das Abspielen von Musik oder das Übertragen von Daten über das Netzwerk.

Content Provider Über einen *Content Provider* lassen sich Daten von Anwendungen (gemeinsam) nutzen. Diese Inhalte liegen beispielsweise im Dateisystem oder in einer Datenbank und können über eine Schnittstelle abgefragt oder sogar modifiziert werden.

Broadcast Reciever Diese Komponente empfängt systemweite Broadcastnachrichten, zum Beispiel, dass der Bildschirm ausgeschaltet oder gedreht wurde oder die Batterie niedrig ist.

Intent Ein *Intent* ist die Beschreibung einer Operation, um andere *Activities* oder *Services* zu starten oder einen *Broadcast* zu senden.

Die oberste Schicht besteht aus MA, welche in Java geschrieben und mit von Google bereitgestellten Werkzeugen übersetzt und zu einem installierbaren Paket zusammengestellt werden. Diese Pakete können dann beispielsweise über den *Google Play Market*[4] veröffentlicht und monetarisiert werden.

Es gibt Werkzeuge wie *FindBugs*[5] oder *Checkstyle*[6], welche auch *Bad Smells* erkennen und unter Umständen beheben können. Diese nehmen aber keinen expliziten Bezug auf SQ. Des weiteren bietet Google Werkzeuge an, die *Bad Smells* finden und zum Teil lösen. Bis auf *Android Lint*[7], welches in die Entwicklungsstudios *Eclipse* und *IntelliJ Idea* integriert ist[8], sind alle Werkzeuge aber manuell auszuführen. Das heißt sie helfen beim Aufspüren von *Bad Smells*, melden aber Verstöße nicht automatisch. Daher werden diese wohl meist erst dann eingesetzt, wenn man bereits bemerkt hat, dass

[2] Libraries
[3] Android Runtime
[4] http://play.google.com/store
[5] http://findbugs.sourceforge.net/
[6] http://checkstyle.sourceforge.net/
[7] http://developer.android.com/tools/debugging/improving-w-lint.html
[8] http://developer.android.com/tools/index.html

Qualitätsanforderungen nicht ausreichend umgesetzt sind. In *Android Lint* besitzen *Bad Smells* ein Attribut *Category*, mit den Werten *Usability, Security, Correctness*, dass man auch als Qualitätsbezug werten kann. Es werden folgende Werkzeuge bereit gestellt:

dmtracedump/traceview Zeigt eine grafische Ausführungsanalyse.

hierarchyviewer Zeigt die Hierarchie von *Views* einer Benutzerobefläche und hilft zu optimieren.

layoutopt Analysiert das Layout der Benutzeroberfläche, um Recheneffizienz zu erhöhen.

systrace Analysiert die Ausführung der Anwendung und hilft Effizienzprobleme zu diagnostizieren.

procstats Zeigt eine Übersicht des Speicherverbrauchs von Prozessen

Zudem gibt es in Android die Möglichkeit *Entwickleroptionen* zu nutzen, zum Beispiel: *Debugging – GPU-Überschneidung* (siehe Smell Overdrawn Pixel in Kapitel 4.19) oder *CPU-Auslastung anzeigen*.

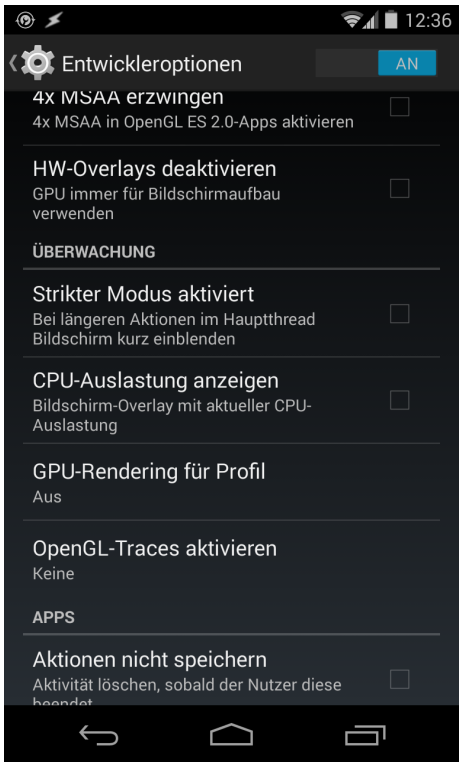


Abbildung 2.2: *Entwickleroptionen unter Android 4.4*

2.2 Qualitäten in der Softwareentwicklung

> Quality is a complex and multifaceted concept.
>
> *(Garvin [Gar84])*

Die Qualität einer Software entscheidet über ihren Erfolg [Lig09]). Das bedeutet, dass Software nicht nur fehlerfrei funktionieren soll, sondern auch, dass sie schnell und bedienerfreundlich sein soll. Doch obwohl es schwer ist, die Qualitäten zu beschreiben und zu definieren, wird in der Anforderungsanalyse versucht, vertraglich zu vereinbaren, wie gut eine Anwendung eine Qualität erfüllen muss. Das ist einerseits wichtig für den Auftraggeber, der eine zufriedenstellende Software bekommen will, andererseits für den Entwickler, der wissen muss, ab wann eine Software *gut genug* ist [Bac96]. Eine Forderung nach beliebig hoher Qualität ist praktisch nicht erfüllbar. Abhilfe schafft hier eine formale Definition von Softwarequalitäten in einem Qualitätsmodell. Ein Beispiel dafür ist in *DIN ISO/IEC 9126*[9] dokumentiert (siehe Abbildung 2.3).

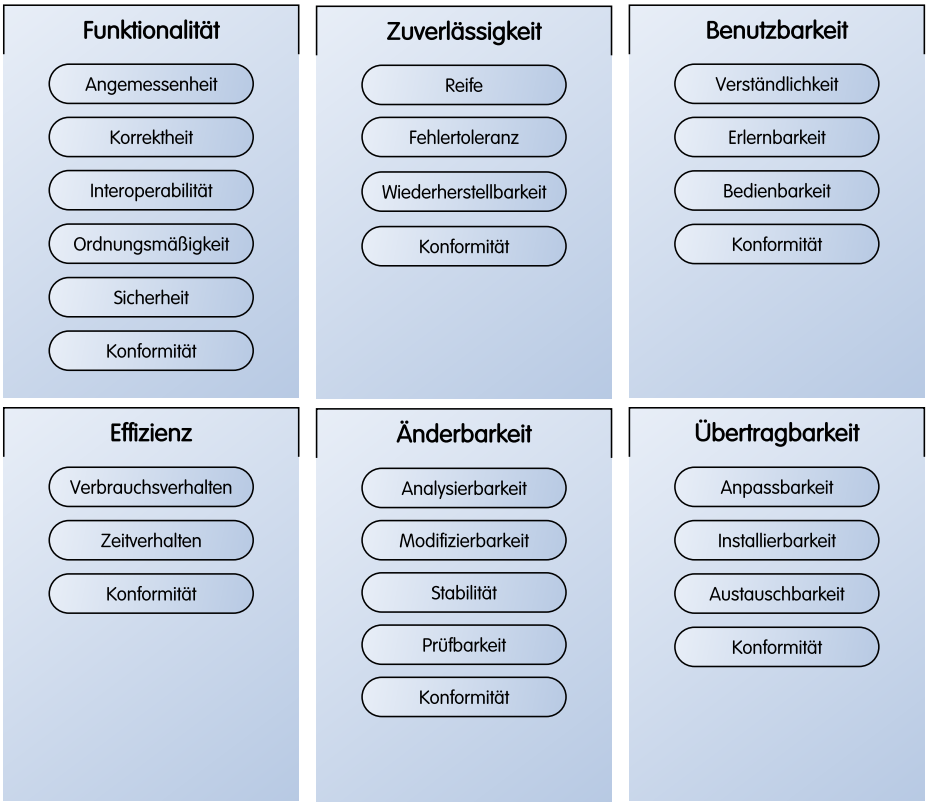


Abbildung 2.3: *Qualitätsmodell nach DIN ISO/IEC 9126 mit den sechs Hauptmerkmalen und vorgeschlagenen Teilmerkmalen*

Eine Eigenschaft dieses Qualitätsmodells ist, dass es hierarchisch aufgebaut ist. Als Beispiel: Der Auftraggeber hat die Anforderung, dass eine Serveranwendung effizient sein soll. Genauer meint er damit, dass sie immer eine festgelegte *Antwortzeit* einhalten

[9] wird zukünftig ersetzt durch Normenkatalog ISO/IEC 25000 – Systems and Software Quality Requirements and Evaluation (SQuaRE)

und auch unter Volllast, einen gewissen *Datendurchsatz* bieten soll. So wird der Oberbegriff Qualität durch Qualitätsmerkmale definiert („Effizienz") und diese wiederum durch Qualitätsteilmerkmale („Zeitverhalten", „Verbrauchsverhalten"). Diese Zerteilung setzt sich fort, bis eine messbare Eigenschaft vorliegt, sogenannte Qualitätsindikatoren („Antwortzeit", „Datendurchsatz") [Geb11]. Da das Modell nicht vollständig ist, muss es zudem fallspezifisch konkretisiert und die vorgeschlagenen Teilqualitäten um weitere ergänzt werden. Zum Beispiel Rentabilität oder Betriebssicherheit und Autonomie (zum Beispiel bei Robotern). Zudem können durch weiteren technologischen Fortschritt andere Softwarequalitäten entstehen oder in den Fokus rücken. Für die Definition und Beschreibung der Benutzerfreundlichkeit kann die Norm *EN ISO 9241* herangezogen werden, welche die Mensch-Computer-Interaktion betrachtet.

Die Messung der Qualitätsindikatoren muss nachprüfbar und reproduzierbar sein (siehe Tabelle 2.1). Indikatoren lassen sich entweder statisch (zeitunabhängig) messen, zum Beispiel die Metrik zur Beurteilung der Programmkomplexität nach McCabe [McC76], oder dynamisch (zeitabhängig), zum Beispiel bei der Beurteilung des Speicherverbrauchs einer Anwendung. Aber auch Testszenarien, zum Beispiel bei der Verständlichkeit einer Dialogführung, können als dynamischer Prozess betrachtet werden. Die Schwierigkeit der Automatisierbarkeit einer Messung ist technologieabhängig. So ist es zur Zeit schwer die Energieeffizienz von Software zu beurteilen. Dies kann sich aber durch technologischen Fortschritt, wie neue Softwareplattformen, zum Beispiel *JouleUnit*, ändern [WGR13]. Generell gilt, dass Metriken schneller zu berechnen und reproduzierbar sind, da sie einen numerischen Wert besitzen. Dahingegen tendieren manuell durchzuführende Messungen dazu aufwendig und nur bedingt reproduzierbar zu sein, da meist nur ein Gutachten entsteht, das selbst interpretiert werden muss. In Tabelle 2.1, welche die qualitative Einordnung der Automatisierbarkeit verdeutlicht, zeigen wir diese Unterscheidung durch die Begriffe „objektiv" und „subjektiv".

Tabelle 2.1: *Qualitative Übersicht über die Messbarkeit von ausgewählten Softwarequalitäten*

AUTOMATISIERBARKEIT	OBJEKTIV	SUBJEKTIV
Schwierigkeit ↑	Energieeffizienz	Verständlichkeit
	Speichereffizienz	Erlernbarkeit
	Recheneffizienz	Bedienbarkeit
	Sicherheit	
	Antwortzeit	Analysierbarkeit
	Korrektheit	Komplexität
	Startzeit	Benutzerkonformität

Metriken von Softwarequalitäten lassen sich auch nach dem Kontext differenzieren. Es gibt zum einen interne Metriken, die nur Eigenschaften innerhalb des Produktes messen (Zeitkomplexität eines Algorithmus). Und zum anderen extrene Metriken, die auch die Interaktion mit anderen Systemen berücksichtigen (Effektivität eines Algorithmus auf einer konkreten Hardware). Die „quality in use metrics" bewerten das Produkt aus der Sicht des Benutzers.

Doch SQ haben noch weitere Aspekte, unter denen sie betrachtet werden können. So stellte Garvin [Gar84] fünf Perspektiven auf SQ vor:

Transzendente Sicht Beschreibt Qualität als nicht präzise definierbar und nur durch Erfahrung erkennbar. Ähnelt Platons Definition von Schönheit.

Produktorientierte Sicht Betrachtet Qualität als präzise messbare Variable (von „atomaren“ Eigenschaften des Produktes), womit sich die Qualität von Produkten vergleichen lässt. Das bedeutet, dass höhere Qualität nur durch höhere Kosten erreicht werden kann und das Qualität einem Produkt inne wohnt und nicht zugeschrieben wird.

Benutzerorientierte Sicht Geht vom Ansatz aus, dass „Schöhnheit im Auge des Betrachters“ liegt. Das heißt, verschiedene Benutzer haben unterschiedliche Anforderungen an ein Produkt, was zu Problemen bei der Auswahl und Zusammenstellung von Qualitäten für bestimmte Benutzer- oder Zielgruppen führt.

Herstellerorientierte Sicht Betrachtet Qualität als Einhaltung von Anforderungen während des Herstellungsprozesses. Das bedeutet das jede Abweichung ein qualitativ schlechteres Produkt bedeutet.

Werteorientierte Sicht Betrachtet Qualität als Kosten-Nutzen-Verhältnis. Damit werden zwei Konzepte vermischt: Qualität als Maß der Exzellenz und Qualität als Maß des Wertes eines Produktes.

Außerdem konnten wir vier Interessengruppen ableiten, welche sich allerdings von *DIN ISO/IEC 9126* unterscheiden:

Softwareentwickler Analysiert, plant, implementiert, testet und wartet das Softwaresystem und legt den Fokus auf Änderbarkeit.

Softwareadministrator Legt den Fokus auf Übertragbarkeit und Zuverlässigkeit.

Anwender Nutzt die Anwendung und legt Wert auf Funktionalität, Benutzbarkeit, Zuverlässigkeit und Effizienz. Aber auch die Frage nach Rentabilität ist wichtig.

Betreiber Ist meist auch der Auftraggeber der Software und legt in erster Linie Wert auf Rentabilität. Diese ist natürlich abhängig vom Erfüllen anderer Qualitäten, die für den Anwender relevant sind. Nur funktionale und qualitativ gute Software verkauft sich weiter. Auch Effizienz kann wichtig werden, wenn eine Client-Server-Infrastruktur bereitgestellt wird, deren Betrieb Geld kostet.

Die Rollen können auch überlappen. So muss der Anwender durchaus auch die Anwendung installieren und administrieren.

Jedoch erfüllt das Qualitätsmodell aus *DIN ISO/IEC 9126* einige Anforderungen nicht ausreichend [Dei09b]:

Exakte Definition Es wird versucht ungenaue Begriffe, wie Änderbarkeit durch weitere ungenaue Begriffe, wie Analysierbarkeit oder Modifizierbarkeit zu definieren. Das führt dazu, dass es schwer ist, diese Qualitäten zu überprüfen, was deren Nutzen in Frage stellt.

Disjunktheit Für ein Qualitätsteilmerkmal sind in der Regel mehrere Qualitätsindikatoren relevant. Zum Beispiel lässt sich die Qualität Speichereffizienz mit den Indikatoren Speicherverbrauch des Programms auf der Festplatte und des Hauptspeichers messen. Zum anderen ist der Speicherverbrauch auf der Festplatte auch für Installierbarkeit und Benutzbarkeit relevant, besonders im mobilen Bereich mit begrenztem Speicherplatz. Damit hat ein Indikator auch Auswirkungen auf mehrere Qualitätsteilmerkmale. Das zeigt, dass diese Merkmale nicht überschneidungsfrei sind. Überlappungen der Bedeutung höherwertigerer SQ sind vermutlich nicht ganz zu vermeiden, jedoch erschwert die unklare Definition vieler Begriffe eine genauere Abgrenzung.

Vollständigkeit Als Beispiel: Die für Softwareentwickler wichtige Eigenschaft der Wiederverwendbarkeit kann nicht direkt in das Modell eingeordnet werden. SQ.

Evaluierbarkeit SQ werden keine konkreten Metriken zugeordnet. Zudem ist nicht geklärt, wie Qualitätsteilmessungen aggregiert werden können, um höherwertigere Aussagen zu treffen.

Abhängigkeit Es ist nicht geklärt, welchen Einfluss Softwarequalitäten aufeinander haben. Tabelle 4.3 lässt Abhängigkeiten vermuten, zum Beispiel erhöht eine Verbesserung der Antwortzeit die Benutzbarkeit, aber nicht umgekehrt.

Deißenböck [Dei09a] hat aus diesem Grund ein alternatives, aktivitätsbasiertes Qualitätsmodell vorgestellt und zeigt es exemplarisch an der Qualität Wartbarkeit. Dabei werden externe Produkteigenschaften (Wartbarkeit) zu internen Produkteigenschaften (Redundanz, Komplexität) zerlegt. In dieses Qualitätsmodell werden mehrere Dimensionen von SQ mit einbezogen (Rolle, Phase, Kosten/Aufwand, Metrik, Artefakt). Sie stellen außerdem eine Werkzeuguntertützung vor, mit denen sich Metriken der Wartbarkeit zu einem Qualitätsleitstand zusammen setzen lassen, der dann zu jeder Zeit Auskunft gibt, wie gut die Software Qualitätsanforderungen erfüllt. Abbildung 2.4 zeigt ein Schema für das aktivitätenbasierte Qualitätsmodell für die SQ Wartbarkeit. Es zeigt die Relation zwischen Qualität, Artefakt, Aktivität und Kosten. So wirkt sich die Strukturiertheit einer Methode positiv auf Impact-Analyse und Änderung aus, eine erhöhte Redundanz jedoch negativ.

Im Allgemeinen jedoch, wird meist erst in der Testphase festgestellt, ob eine Software hinsichtlich einer Qualität optimiert werden muss. Da ein Qualitätsaspekt, das ganze System durchziehen kann, ist dieser Prozess allerdings komplex und fehleranfällig [MTM07]). Um den Aufwand der Optimierung gering zu halten, wird versucht schon in früheren Phasen gute Software zu produzieren. Darum gibt es einen Bedarf an ingenieurstechnischen Lösungen für wiederkehrende Probleme (*Bad Smells*). Solche Kataloge mit Musterlösungen (*Patterns*) gibt es für verschiedene Bereiche des Softwareengineering:

Organisation

- *AntiPatterns: Refactoring Software, Architectures, and Projects in Crisis* von Brown [Bro98]

Anforderungsanalyse

- *Use cases: patterns and blueprints* von Övergaard und Palmkvist [ÖP05]

Softwarearchitektur

- *Patterns of Enterprise Application Architecture* von Fowler [Fow99]

				Wartbarkeit			
				Analyse		Implementierung	
				Concept Location	Impact Analyse	Erstellung	Änderung
Produkt	Software	Klasse	Strukturiertheit	+	+		+
		Methode	Strukturiertheit		+		+
			Redundanz		-		-
	Dokumentation	Glossar	Vollständigkeit	+		+	
		Kommentar	Prägnanz	+		+	+

Abbildung 2.4: *Aktivitätenbasiertes Qualitätsmodell*

- *Design Patterns: Elements of Reusable Object-Oriented Software* von Gamma u. a. [Gam+94]
- *Identifying Architectural Bad Smells* von Garcia u. a. [Gar+09]

Implementierung

- *Refactoring: Improving the Design of Existing Code* von Fowler u. a. [Fow+12]
- *New Software Performance AntiPatterns: More Ways to Shoot Yourself in the Foot* von Smith und Williams [SW03]
- *Implementation Pattern* von Beck [Bec07]

Doch gerade Veröffentlichungen über Softwaredesign und -implementierung behandeln *Bad Smells* und *Refactorings* lediglich aus der Perspektive der Softwareentwickler. So definiert Fowler u. a. [Fow+12] ein *Refactoring* folgendermaßen: „a change made to the internal structure of software to make it easier to understand and cheaper to modify without changing its observable behavior." und beschränkt die betrachteten Qualitäten damit auf Analysierbarkeit und Modifizierbarkeit. Auch der Trend zur *Green IT* umfasst ausschließlich die Energieeffizienz und viele Arbeiten beschäftigen sich nur mit der Verbesserung von Anwendung hinsichtlich deren Verbrauch, zum Beispiel: [HB10; Got+12; Pat+12; Vet+13].

Das zeigt, dass Entwickler das Wissen um Qualitäten beim Identifizieren und Refaktorieren nur implizit besitzen und anwenden. Es fehlt jedoch der generelle Bezug von *Bad Smells* zu Softwarequalitäten, was das Optimieren von Software hinsichtlich einer speziellen Qualität behindert. So lassen bisher auch Werkzeuge SQ unbeachtet, doch gerade das Empfehlen konkreter *Refactorings*, ist eine erwünschte Eigenschaft, wie eine Untersuchung von Pinto und Kamei [PK13] zeigt.

2.3 Quality Smells

Um den genannten Nachteilen zu begegnen wird von Reimann und Aßmann [RA13] vorgeschlagen, dass Konzept der *Bad Smells* und des *Refactorings* mit dem Aspekt der

Softwarequalitäten zu verbinden. Eine Übersicht, in UML, über die Beziehung zwischen *Quality Smells*, *Refactorings* und Qualitäten, ist in Abbildung 2.5 zu sehen. Dabei zeichnet sich ein *Quality Smell* dadurch aus, dass es mindestens eine SQ negativ beeinflusst und mehrere *Refactorings* besitzen kann, die wiederum mindestens eine dieser SQ positiv beeinflusst. Diese explizite Relation ist ein wichtiger Schritt in Richtung der qualitätsorientierten Softwareentwicklung. Sie erlaubt es, während der Optimierung den Fokus auf einzelne SQ zu legen und nur beeinflussende *Bad Smells* zu identifizieren und refaktorieren.

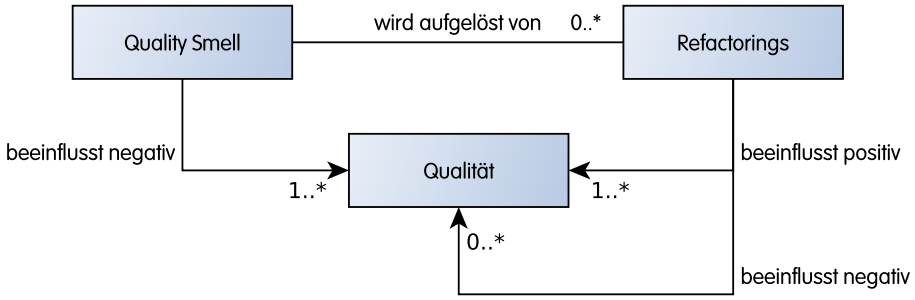


Abbildung 2.5: *Beziehung* Quality Smell*,* Refactoring *und Qualität*

Quality Smells haben notwendige Bedingungen, die statisch feststellbar sind, und hinreichende Bedingungen, die dynamisch feststellbar sind. Als Beispiel soll hier der *Quality Smell* Leaking Inner Class (Kapitel 4.12) dienen: Statisch bedeutet, dass es beispielsweise durch Quelltextanalyse möglich ist, ein potentielles Speicherleck zu finden, indem nach inneren Klassen vom Typ `Handler` gesucht wird. Die hinreichende Bedingung, dass es auch wirklich zu einem Leck führt, lässt sich aber nur dynamisch fest stellen, zum Beispiel durch das Untersuchen eines zur Laufzeit erstellten Speicherabbildes.

Neue Anforderungen und Technologien, sowie zu hohe Komplexität, durch technische und organisatorische Fehlentscheidungen kann Software altern [Par94; Leh80]. Das kann dazu führen, dass Qualitätsprobleme nicht mehr allein durch *Refactoring* einzelner Artefakte zu beheben sind. Dann kann es notwendig sein, Teile oder das ganze System zu renovieren [ASH13; Dei09a]. Jedoch lässt sich durch den frühzeitigen Einsatz geeigneter Werkzeuge, die Geschwindigkeit der Softwarealterung verringern.

3 Studie

> In our experience no set of metrics rivals informed human intuition
>
> *(Fowler [Fow99])*

3.1 Durchführung

Ziel der Studie ist das Identifizieren von *Quality Smells* und deren Lösung. Dabei soll auf das Wissen von Programmierern zurück gegriffen werden, da diese eine zentrale Rolle im Entwicklungsprozess einnehmen und über implizites Wissen verfügen, MA hinsichtlich SQ zu optimieren. Um an diese Informationen zu gelangen, kommen mehrere Arten von Studien in Frage:

1. Beobachten bei der Entwicklung
2. Interviews und Umfragen
3. Aggregation von Dokumenten, wie Quellcode, Dokumentation, Bug-Tracker und Foren

Studie (1) hat den Vorteil, dass man konkret sieht, wie der Entwickler programmiert, *Quality Smells* feststellt und beseitigt. Allerdings ist diese Vorgehensweise sehr zeitaufwendig und auf die Funktionalität der Anwendung beschränkt.

Studie (2) ist weniger zeitaufwendig und spricht mehrere Entwickler an. Allerdings ist es möglich, dass unbewusstes Wissen nicht abgerufen oder firmeninternes nicht weiter gegeben wird und *Quality Smells* somit nicht entdeckt werden.

Studie (3) ist zeitaufwendiger als (2), da viele Informationen gesammelt und ausgewertet werden müssen, allerdings wird auf diese Weise auch vergangenes Wissen durchsucht.

Bei der Entscheidung für die Art der Studie spielte auch der Open-Source-Charakter Androids eine große Rolle. So tragen viele kostenlose Internetplattformen, wie Bug-Tracker, Versionsverwaltungssysteme, Foren und Hilfen, dazu bei, dass Entwickler miteinander an Programmen und Bibliotheken arbeiten und ihre Erfahrungen untereinander austauschen. Somit ist die Entscheidung auf Studie (3) und das Durchsuchen von Informationen im Internet gefallen.

Die folgenden Internetplattformen wurden aufgrund ihrer Popularität und Bedeutsamkeit für Androidentwickler ausgewählt:

Stackoverflow[1] Gehört zum Stackexchange-Netzwerk, der populärsten Community für Entwickler[2]. Es ist ein Forum im Frage & Antwort-Stil mit Reputationssystem, das heißt sowohl Fragen, als auch Antworten können bewertet werden.

[1] http://stackoverflow.com

[2] http://www.alexa.com/siteinfo/stackoverflow.com

Programmers Stackexchange[3] Gehört zum Stackexchange-Netzwerk und befasst sich im konkreten Sinne mit Programmierung.

Android Stackexchange[4] Gehört zum Stackexchange-Netzwerk und befasst sich mit Androids gesamten Ökosystem.

Android Issues[5] Ist der offizielle Bug-Tracker der Android Plattform.

Android Developers in Google Groups[6] Ist ein offizielles Google-Forum für Android Entwickler.

Android Developers Blogspot[7] Ist ein bekannter und offizieller Blog von Google Mitarbeitern.

Android Design Patterns[8] Ein anderer bekannter Blog.

Android Entwicklerdokumentation[9] Die Entwicklerdokumentation für Android von Google, ist der typische Anlaufpunkt zum Erlernen und Nachschlagen der API.

Google IO Vorträge[10] Googles Vortragsreihe über verschiedene Themen spricht weltweit viele Entwickler an, sich über Neuerungen und Best Practices zu informieren und auszutauschen.

Pro Android Apps Performance Optimization Ist ein Buch von Guihot [Gui12].

Durch die enorme Datenmenge erfolgte das Durchsuchen der Webplattformen bis Juli 2013 über bis zu drei Phasen (Abbildung 3.1):

1. Herunterladen aller Einträge in eine lokale Datenbank
2. Filtern der Einträge durch spezielle Schlagworte, siehe Anhang A.1. Bei der Auswahl der Filterworte, wurde der Fokus auf Effizienz (Energie, Speicher, Zeit) und die Andeutung von *Bad Smells* gelegt.
3. Manuelle Suche in gefilterteten Einträgen, das bedeutet Durchlesen der Einträge, sowie verfolgen von Referenzen

Die Seiten des *Stackexchange*-Netzwerkes können über eine SQL-Schnittstelle[11] abfragt werden. Damit lassen sich Erhebung und Filterung kombinieren. Mit 2,7 Millionen Nutzern und über 6,3 Millionen Fragen, sowie 11 Millionen Antworten, allein auf *stackoverflow.com*[12], musste der Abfragezeitraum allerdings aufgeteilt werden, da es sonst zu Zeitüberschreitungen auf dem Server kam. So wurden die 500 bestbewerteten Fragen

[3] http://programmers.stackexchange.com
[4] http://android.stackexchange.com
[5] http://issues.android.com
[6] http://groups.google.com/d/forum/android-developers
[7] http://android-developers.blogspot.de
[8] http://www.androiddesignpatterns.com/
[9] http://developer.android.com
[10] https://developers.google.com/events/io/
[11] http://data.stackexchange.com
[12] http://stackexchange.com/sites

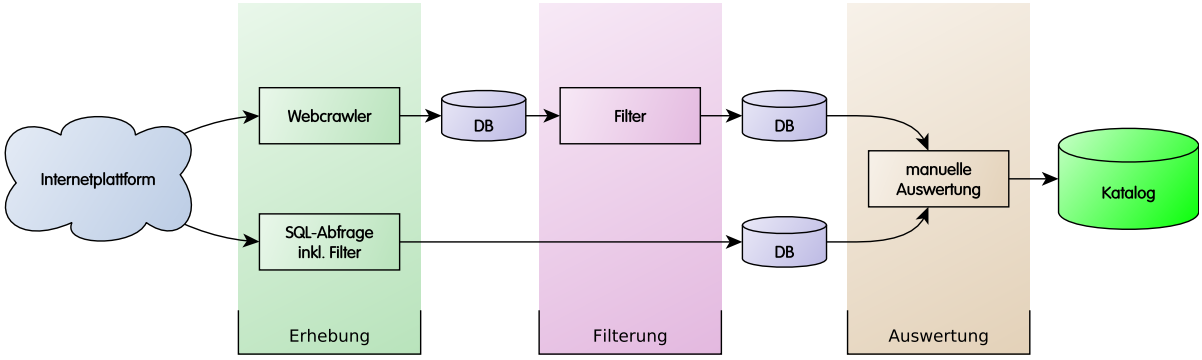


Abbildung 3.1: *Schema der Websuche*

von drei Zeiträumen (bis 2010, 2010 bis 2012, 2012 bis Juli 2013) abgefragt und in eine lokale Datenbank gespeichert.

Da Android erst 2008 auf einem Gerät erschien und seitdem viele Änderungen an der API erfolgten, gibt es bis 2010 allerdings keine nennenswerten und noch relevanten Einträge. So ist die Android Version 2.2 „Froyo“, erschienen im Mai 2010, nur noch mit 1,7% Marktanteil vertreten[13].

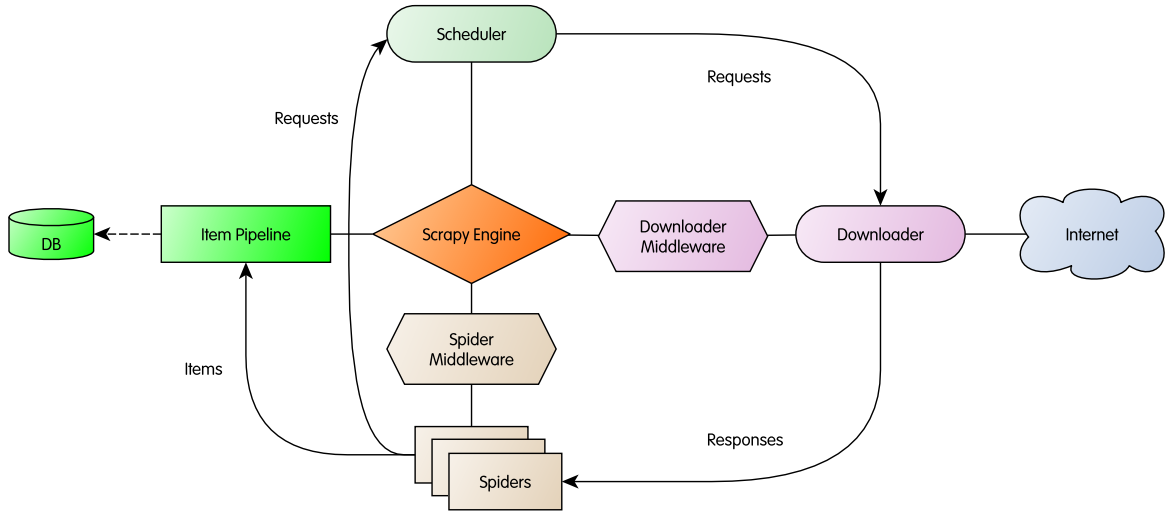


Abbildung 3.2: *Architektur von Scrapy*

Für die Seiten *Android Issues* und *Android Developers in Google Groups* wurde je ein Webcrawler geschrieben, der automatisch alle Einträge herunterlädt, aufbereitet und in eine *SQLite*-Datenbank importiert. Dafür wurde das *Python*-Framework *Scrapy*[14] genutzt. In Abbildung 3.2 ist die Architektur von Scrapy skizziert. Hauptbestandteil sind:

Item Repräsentiert ein einzelnes Ergebnis (Beitrag, Frage, Antwort).

Spider Analysiert Webseiten beispielsweise über einen XPath-Ausdruck und generiert und leitet *Items* an die *Engine* weiter.

[13] http://de.statista.com/statistik/daten/studie/180113/
[14] http://www.scrapy.org

Item Pipeline Verarbeitet *Items* weiter (speichert sie zum Beispiel in eine Datenbank).

Scheduler Empfängt *Requests* von der *Engine* und bearbeitet sie nach einem festgelegten Schema.

Downloader Lädt Webseiten herunter.

Engine Empfängt *Items* und *Requests* von einer *Spider*, sowie heruntergeladene Webseiten vom *Downloader* und leitet diese an *Item Pipeline* und *Scheduler*, beziehungsweise die *Spider* weiter.

Downloader Middleware Bieten einen Weg die *Downloader*-Funktionalität mit eigenem Code zu erweitern. Wird unter anderem verwendet, um den User-Agent zu verändern, Cookies zu setzen oder sich zu authentifizieren.

Spider Middleware Bieten einen Weg die *Spider*-Funktionalität mit eigenem Code zu erweitern. Wird unter anderem verwendet, um festzustellen, wie weit entfernt der momentane vom Start-*Request* ist, um so eine maximale Tiefe festzulegen.

3.2 Schema eines Quality Smells

Um bei der manuellen Suche die identifizierten *Quality Smells* zu katalogisieren, wurden sie in folgendes Schema konvertiert:

Name Name	ein eindeutiger Name
Betroffene Qualitäten Affected Qualities	eine Liste von SQ, die negativ beeinflusst werden.
Kontext Context	eine anwendungstypische Einordnung des *Quality Smells*, zum Beispiel UI, Netzwerk, Sensoren.
Stichworte Tags	eine Liste von Schlagworten, zur weiteren Einordnung und Berechnung von *Verwandten Smells*.
Beschreibung Description	eine detaillierte Beschreibung des Problems, wenn möglich mit Beispiel.
Lösungen Refactorings	eine Liste von Lösungen/Refactorings, mit den folgenden Attributen:
Name Name	ein eindeutiger Name
Aufgelöste Qualitäten Resolved Qualities	eine Liste von SQ, die durch diese Lösung positiv beeinflusst werden.
Betroffene Qualitäten Affected Qualities	eine Liste von SQ, die durch diese Lösung negativ beeinflusst werden.
Beschreibung Description	eine detaillierte Beschreibung der Lösung, wenn möglich mit Beispiel.
Referenzen References	ein Liste von Referenzen, wie Quelle oder weitergehende Dokumentation.
Verwandte Smells Related Smells	eine List verwandter *Quality Smells*, die sich aus den *Stichworten* berechnen.

Dabei wurden die im Katalog verwendeten englischen Begriffe darunter geschrieben. Das Schema lehnt sich dabei an die Arbeiten von Fowler u. a. [Fow+12] und Gamma u. a. [Gam+94], sowie dem *Portland Pattern Repository* von Ward Cunningham[15] und den *Anti Patterns* von Brown [Bro98] an. Die Beschreibung des *Refactorings* vereint dabei die Punkte *Motivation, Mechanics, Examples* von Fowler.

3.3 Ergebnisse

Tabelle 3.1 zeigt eine Übersicht über die Anzahl gefilterter Einträge und darin gefundener *Quality Smells*. Zu Beachten ist, dass ein *Quality Smell* öfter dokumentiert worden sein kann, aber nur bei einer Quelle angerechnet worden ist. Unter die Kategorie *Sonstige*

Tabelle 3.1: *Anzahl gefundener* Quality Smells

QUELLE	NACH FILTERUNG	GEFUNDENE SMELLS
Stackoverflow	1116	3
Programmers Stackexchange	28	1
Android Stackexchange	500	0
Android Issues	466	0
Android Developer in Google Groups	556	1
Android Entwicklerdokumentation	*	2
Android Developers Blog	*	5
Android Design Patterns	*	2
Google IO Vorträge	*	8
Pro Android Apps Performance Optimization	*	5
Sonstige	*	3

* nur manuell durchsucht

fallen Einträge, die durch bisherige bekannte Arbeiten oder Kreuzreferenzen gefunden wurden.

Der Katalog ist in Kapitel 4 zu finden.

3.4 Aus- und Bewertung

Durch das Durchsuchen von Webplattformen sollten *Quality Smells* gefunden werden, welche dann in einen Katalog einzugliedern sind. Das Ergebnis von 30 Einträgen mit 9 beeinflussten Qualitäten zeigt, dass qualitätsbewusste Softwareentwicklung angestrebt wird, aber auch, dass durch die hohe Komplexität der Android-Plattform Fehler gemacht werden und damit ein Bedarf an Best Practices und das Erkennen und Auflösen von *Quality Smell* besteht. Der Katalog bietet somit eine schnelle und einfache Möglichkeit bekannte Probleme nachzuschlagen und somit zu erkennen und nötigenfalls zu beheben. Aus dem Ergebnis lässt sich jedoch nicht ableiten, wie groß der Anteil an Softwareentwicklern ist, welche Qualitätsprobleme haben, entdecken und versuchen aufzulösen.

Durch die Art der durchsuchten Webplattformen, die sich eher mit der Programmierung beschäftigt, ergibt sich, dass die meisten *Smells* in eben diese Kategorie passen. Es ist anzunehmen, dass durch die Ausweitung der Filterwörter auf andere Softwarequalitäten und das Durchsuchen weiterer Plattformen, zum Beispiel zum Thema Benutzbarkeit, noch mehr *Quality Smells* finden lassen.

Die Anzahl gefundener *Quality Smells* (siehe Tabelle 3.1) zeigt, dass die Mehrzahl in der *Android Entwicklerdokumentation* und dem *Android Developer Blog* von Google, sowie in *Google IO Vorträgen* zu finden waren. Auch verweisen viele Einträge auf diese Plattformen. Das kann zum einen daran liegen, dass Google auf berichtete Probleme von anderen Entwicklern reagiert hat, andererseits auf eigene Erfahrung bei der Entwicklung von MA (*Google Play*, *Google Calendar*) zurück greifen konnten. Das zeigt, dass das Be-

[15] http://c2.com/cgi/wiki?AntiPattern

wusstsein hochqualitative Anwendungen zu erstellen, vorhanden ist und Informationen bereit gestellt werden.

Die Anzahl an Softwarequalitäten mit Bezug zu Benutzbarkeit zeigt, dass sich am ehesten aus dem langsamen Verhalten der Benutzeroberfläche auf Qualitätsprobleme schließen lässt. So war auch der am häufigsten zu beobachtende *Quality Smell* Uncached Views (Kapitel 4.26) mit dem *Refactoring* INTRODUCE VIEW HOLDER. Jedoch wurden auch mehrere Speicherprobleme (*memory leaks*) festgestellt, zu welchen allerdings selten eine Lösung dokumentiert wurde. Das kann daran liegen, dass die Ursache von *memory leaks* schwer zu identifizieren ist. Bei *Android Issues* wurden größtenteils Fehler dokumentiert, die aufgrund einer neueren Androidversion nicht mehr behoben wurden. Auf der Seite *Android Stackexchange* gibt es viele Hinweise auf Qualitätsprobleme aus Benutzersicht, aber es werden keine konkreten Lösungen für Entwickler dokumentiert.

3.4.1 Validität

Die Sammlung der *Quality Smells* ist nicht vollständig. Zum einen wurde nur eine Auswahl bekannter Webplattformen durchsucht, zum anderen konnten auch nur Informationen analysiert werden, die öffentlich dokumentiert wurden, weil ihre Urheber ihr Wissen teilen. Durch die Auswahl der Filterwörter, aber auch die große Menge an Daten nach der Filterung, die manuell überprüft werden mussten, ist es zudem möglich, dass nicht alle *Quality Smells* entdeckt worden sind. Zu beachten ist auch, dass die SQ der *Bad Smells* und *Refactorings* so wiedergegeben wurden, wie in den Forendiskussionen, Vorträgen und Ähnlichen berichtet. Das bedeutet, dass keine exakte, reproduzierbare Messung zu Grunde liegen muss. Jedoch geben einige Quellen auch Beweise für die Optimierung an.

4 Katalog

If it stinks, change it.

(Fowler [Fow99])

Englisch ist die *Lingua franca* der Computerwissenschaft und damit der Softwareentwicklung, so waren auch alle untersuchten Webplattformen in dieser Sprache. Aus diesem Grund wird der Katalog auch in Englisch veröffentlicht.

Eine Online-Version wird unter folgender Adresse bereitgestellt:

http://www.modelrefactoring.org/smell_catalog/

Table 4.1: *Listing of* Quality Smells *and their context*

Smell	Database	IO	Implement	Network	UI
Bulk Data Transfer On Slow Network				×	
Data Transmission Without Compression			×	×	
Debuggable Release			×		
Dropped Data					×
Durable WakeLock			×		×
Early Resource Binding			×		×
Inefficient Data Structure			×		
Inefficient SQL Query	×		×	×	
Inefficient Data Format And Parser		×		×	
Internal Getter/Setter			×		
Interrupting From Background					×
Leaking Inner Class			×		
Leaking Thread			×		
Member-Ignoring Method			×		
Nested Layout					×
Network & IO Operations In Main Thread		×	×	×	×
No Low Memory Resolver			×		
Not Descriptive UI					×
Overdrawn Pixel					×
Prohibited Data Transfer				×	
Public Data			×		
Rigid AlarmManager			×		
Set Config Changes			×		
Slow Loop			×		
Tracking Hardware Id			×		
Uncached Views					×
Unclosed Closable			×		
Uncontrolled Focus Order					×
Unnecessary Permission			×		
Untouchable					×

Table 4.3: *Listing of* Quality Smells *and their affected qualities*

Smell	Accessibi	Efficiency	Energy Ef	Memory F	Security	Stability	Startup T	User Con	User Experience
Bulk Data Transfer On Slow Network			×					×	
Data Transmission Without Compression			×						
Debuggable Release					×				
Dropped Data								×	×
Durable WakeLock			×						
Early Resource Binding			×						
Inefficient Data Format And Parser		×							
Inefficient Data Structure		×							
Inefficient SQL Query		×							
Internal Getter/Setter		×							
Interrupting From Background								×	×
Leaking Inner Class				×					
Leaking Thread				×					
Member-Ignoring Method		×							
Nested Layout		×					×		×
Network & IO Operations In Main Thread									×
No Low Memory Resolver				×		×			×
Not Descriptive UI	×							×	×
Overdrawn Pixel		×							
Prohibited Data Transfer			×					×	
Public Data					×			×	
Rigid AlarmManager		×	×						
Set Config Changes				×				×	
Slow Loop		×							
Tracking Hardware Id					×			×	
Uncached Views		×							
Unclosed Closable				×					
Uncontrolled Focus Order	×								×
Unnecessary Permission					×			×	
Untouchable	×								×

4.1 Bulk Data Transfer On Slow Network

AFFECTED QUALITES
energy efficiency, user conformity
CONTEXT
network
TAGS
network, energy, user conformity
DESCRIPTION
According to a Google IO presentation[1] transferring data over a slower network connection will consume much more power then over a fast connection.

Transferring a 6 MB file costs:

$$\begin{aligned} \textbf{EDGE (90kbps):} \quad & 300\,\mathrm{mA} \times 9.1\,\mathrm{min} && = 44\,\mathrm{mAh} \\ \textbf{3G (300kbps):} \quad & 210\,\mathrm{mA} \times 2.7\,\mathrm{min} && = 9.5\,\mathrm{mAh} \\ \textbf{WiFi (1Mbps):} \quad & 330\,\mathrm{mA} \times 48\,\mathrm{s} && = 4.4\,\mathrm{mAh} \end{aligned}$$

Refactorings

Check network connection

RESOLVED QUALITIES
energy efficiency, user conformity
AFFECTED QUALITIES
-
DESCRIPTION
It should be first checked which network connection we are currently using.

Code 4.1.1: Connection Check

```
ConnectivityManager mConnectivity;
TelephonyManager mTelephony;
// Skip if no connection, or background data disabled
NetworkInfo info = mConnectivity.getActiveNetworkInfo();
if (info == null || !mConnectivity.getBackgroundDataSetting()) {
    return false;
}
// Only update if WiFi or 3G is connected and not roaming
int netType = info.getType();
int netSubtype = info.getSubtype();
if (netType == ConnectivityManager.TYPE_WIFI) {
    return info.isConnected();
} else if (netType == ConnectivityManager.TYPE_MOBILE
    && netSubtype == TelephonyManager.NETWORK_TYPE_UMTS
```

[1] http://www.google.com/events/io/2009/sessions/CodingLifeBatteryLife.html

```
    && !mTelephony.isNetworkRoaming()) {
    return info.isConnected();
} else {
    return false;
}
```

Another approach is to give the user the choice (user preference), when high volume data transfer should be done, eg. only WiFi, 3G etc. A manual trigger of the update process might also be a solution.

References

- http://www.google.com/events/io/2009/sessions/CodingLifeBatteryLife.html
- http://developer.android.com/reference/android/net/ConnectivityManager.html

Related Smells

- Data Transmission Without Compression (4.2)
- Durable WakeLock (4.5)
- Early Resource Binding (4.6)
- Inefficient Data Format And Parser (4.7)
- Inefficient SQL Query (4.9)
- Interrupting From Background (4.11)
- Not Descriptive UI (4.18)
- Network & IO Operations In Main Thread (4.16)
- Prohibited Data Transfer (4.20)
- Public Data (4.21)
- Rigid AlarmManager (4.22)
- Set Config Changes (4.23)
- Tracking Hardware Id (4.25)
- Unnecessary Permission (4.29)

4.2 Data Transmission Without Compression

Affected Qualites
energy efficiency
Context
implementation, network
Tags
network, energy
Description
Höpfner and Bunse discussed in their publication „Towards an energy-consumption based complexity classification for resource substitution strategies“ that transmitting a file over a network infrastructure without compressing it consumes more energy. More precisely, energy consumption is reduced in case the data is compressed at least by 10%, transmitted and decompressed at the other network node.

The following example show file transmission implemented with the Apache HTTP Client Library[2] (the example is taken from ClientMultipartFormPost.java).

Code 4.2.1: Transmision without Compression

```
public static void main(String[] args) throws Exception {
  if (args.length != 1)  {
    System.out.println("File path not given");
    System.exit(1);
  }
  HttpClient httpclient = new DefaultHttpClient();
  try {
    HttpPost httppost = new HttpPost("http://some.url:8080/servlets-
    examples/servlet/RequestInfoExample");
    // the passed File object in this constructor is transmitted
    without compression
    FileBody bin = new FileBody(new File(args[0]));
    StringBody comment = new StringBody("A binary file of some kind");

    MultipartEntity reqEntity = new MultipartEntity();
    reqEntity.addPart("bin", bin);
    reqEntity.addPart("comment", comment);

    httppost.setEntity(reqEntity);

    System.out.println("executing request " + httppost.getRequestLine
    ());
    HttpResponse response = httpclient.execute(httppost);
    HttpEntity resEntity = response.getEntity();

    System.out.println("-----------------------------------------");
    System.out.println(response.getStatusLine());
    if (resEntity != null) {
      System.out.println("Response content length: " + resEntity.
    getContentLength());
```

[2] http://apache.imsam.info//httpcomponents/httpclient/binary/httpcomponents-client-4.2.4-bin.zip

```
    }
    EntityUtils.consume(resEntity);
  } finally {
    try {
      httpclient.getConnectionManager().shutdown();
    } catch (Exception ignore) {
    }
  }

}
```

Refactorings

Add Data Compression to Apache HTTP Client based file transmission

RESOLVED QUALITIES
energy efficiency
AFFECTED QUALITIES
-
DESCRIPTION
Compress the file object before transmitting it.

Code 4.2.2: Transmision with Compression

```
public static void main(String[] args) throws Exception {
  // ...
  // the passed File object now is compressed
  FileBody bin = new FileBody(gzipFile(file)) ;
  // ...
}
private static File gzipFile(File uncompressedFile){
  try {
    assert uncompressedFile != null && uncompressedFile.exists();
    File gzippedFile = File.createTempFile(uncompressedFile.getName(),
     "gz");
    FileInputStream fis = new FileInputStream(uncompressedFile);
    GZIPOutputStream out = new GZIPOutputStream(new FileOutputStream(
   gzippedFile));
    byte[] buffer = new byte[4096];
    int bytesRead;
    while ((bytesRead = fis.read(buffer)) != -1){
      out.write(buffer,0, bytesRead);
    }
    fis.close();
    out.close();
    return gzippedFile;
  } catch (FileNotFoundException e) {
    e.printStackTrace();
  } catch (IOException e) {
    e.printStackTrace();
```

```
    }
    return null;
}
```

References

- **Höpfner, Bunse (2010)** - Towards an energy-consumption based complexity classification for resource substitution strategies
  http://ceur-ws.org/Vol-581/gvd2010_7_1.pdf

Related Smells

- Bulk Data Transfer On Slow Network (4.1)
- Durable WakeLock (4.5)
- Early Resource Binding (4.6)
- Inefficient Data Format And Parser (4.7)
- Inefficient SQL Query (4.9)
- Network & IO Operations In Main Thread (4.16)
- Prohibited Data Transfer (4.20)
- Rigid AlarmManager (4.22)

4.3 Debuggable Release

Affected Qualites
security
Context
implementation
Tags
security, debug
Description
The attribute `android:debuggable` in the `AndroidManifest.xml` is set at development time to debug the app. But it is left in release time. This is a severe security issue.

Refactorings

Remove Debuggable Attribute

Resolved Qualities
security
Affected Qualities
-
Description
Remove the `android:debuggable` attribute or set it to false explicitly.

References

- https://developers.google.com/events/io/2012/sessions/gooio2012/107/
- http://stackoverflow.com/questions/4580595/
- http://developer.android.com/guide/topics/manifest/manifest-intro.html

Related Smells

- Public Data (4.21)
- Tracking Hardware Id (4.25)
- Unnecessary Permission (4.29)

4.4 Dropped Data

Affected Qualites
user experience, user conformity
Context
UI
Tags
user experience, user conformity
Description
The user can input or edit data in an *Activity* or *Fragment*. Imagine another *Activity* pops up (eg. an incoming phone call) and interrupts the user. After returning to the the former *Activity* the input is lost, but the user expects the data to be persisted.

Refactorings

Save instance state

Resolved Qualities
user experience, user conformity
Affected Qualities
-
Description
The developer has to ensure that the state of the *Activity* or *Fragment* is stored, when the user entered data. This is usually done in `onSaveInstanceState(Bundle)`. It can be restored by overriding `onRestoreInstanceState(Bundle)`. For default widgets this is already done by the framework, so do not miss to call

- `super.onCreate(Bundle);`
- `super.onSaveInstanceState(Bundle);`
- `super.onRestoreInstanceState(Bundle);`

in the corresponding methods.

References

- http://stackoverflow.com/questions/151777/
- http://developer.android.com/guide/practices/seamlessness.html#drop
- http://developer.android.com/reference/android/app/Activity.html

Related Smells

- Bulk Data Transfer On Slow Network (4.1)
- Durable WakeLock (4.5)
- Interrupting From Background (4.11)
- Not Descriptive UI (4.18)
- Prohibited Data Transfer (4.20)
- Public Data (4.21)

- Set Config Changes (4.23)
- Tracking Hardware Id (4.25)
- Unnecessary Permission (4.29)

4.5 Durable WakeLock

AFFECTED QUALITES
energy consumption
CONTEXT
ui, implementation
TAGS
ui, wakelock, energy
DESCRIPTION

A WakeLock is needed to tell the system that the app needs to stay the device on, (to use CPU, Sensors, GPS, Network, etc.). After using the ressources the application should release the WakeLock. If this is not done this will drain the battery.

A WakeLock is aquired by:

Code 4.5.1: Aquiring a WakeLock

```
PowerManager pm = (PowerManager) mContext.getSystemService(Context.
    POWER_SERVICE);
PowerManager.WakeLock wl = pm.newWakeLock(
  PowerManager.SCREEN_DIM_WAKE_LOCK
  | PowerManager.ON_AFTER_RELEASE,
  TAG);
wl.acquire();
// ... do work...
wl.release();
```

Refactorings

Aquire WakeLock with timeout

RESOLVED QUALITIES
energy consumption
AFFECTED QUALITIES
-
DESCRIPTION
To ensure that the WakeLock will be released in all circumstances it is recommended to use the method `PowerManager.WakeLock.aquire(long timeout)`.

Code 4.5.2: Aquiring a WakeLock with Timeout

```
PowerManager pm = (PowerManager) mContext.getSystemService(Context.
    POWER_SERVICE);
PowerManager.WakeLock wl = pm.newWakeLock(
  PowerManager.SCREEN_DIM_WAKE_LOCK
  | PowerManager.ON_AFTER_RELEASE,
  TAG);
wl.acquire(60*1000*10); // auto release it in 10 minutes
// ... do work...
```

```
wl.release();
```

References

- *Pro Android Apps Performance Optimization* by [Gui12]
- `http://developer.android.com/reference/android/os/PowerManager.WakeLock.html`

Related Smells

- Bulk Data Transfer On Slow Network (4.1)
- Data Transmission Without Compression (4.2)
- Dropped Data (4.4)
- Early Resource Binding (4.6)
- Prohibited Data Transfer (4.20)
- Rigid AlarmManager (4.22)

4.6 Early Resource Binding

Affected Qualites
energy efficiency
Context
implementation, ui
Tags
energy, location, gps
Description
[Got+12] discussed in „Towards an energy-consumption based complexity classification for resource substitution strategies“ that if physical, energy-consuming resources of an Android device are requested too early more energy is consumed. More precisely, „too early“ means when they are requested in methods being executed before the user is interacting with the app.

For instance, in the following example the GPS component of an Android device is requested in the `onCreate()` method already. Thus, the GPS physical component consumes energy while the user isn't interacting with any map since nothing is visible yet.

This example is taken from the above mentioned publication:

Code 4.6.1: Early Resource Binding

```
public class GpsPrint extends Activity implements OnClickListener,
    Listener, LocationListener {
  public void onCreate(Bundle savedInstanceState) {
    LocationManager lm = (LocationManager) this.getSystemService(
    Context.LOCATION_SERVICE);
    if(lm.getAllProviders().contains(LocationManager.GPS_PROVIDER)){
      if(lm.isProviderEnabled(LocationManager.GPS_PROVIDER)){
        lm.addGpsStatusListener(this);
        // here the physical component is requested and consumes
    energy
        // but the user cannot yet interact with the app in this state
        lm.requestLocationUpdates(LocationManager.GPS_PROVIDER, 1000,
    0, this);
        statusView.setText(?GPS service started?);
      } else {
        statusView.setText(?Please enable GPS?);
        saveLocationButton.setEnabled(false);
      }
    }
  }
  public void onPause() {
    lm.removeUpdates(this);
  }
}
```

Refactorings

Move Resource Request to Visible State Method

RESOLVED QUALITIES
energy consumption
AFFECTED QUALITIES
-
DESCRIPTION
Move the physical resource requesting statement to the `onResume()`. This method corresponds to the visible state of an Activity. Thus, the physical resource is consuming energy but only when the app is visible. Hence, less energy is consumed because of less time.

Code 4.6.2: Delayed Resource Binding

```
public class GpsPrint extends Activity implements OnClickListener,
    Listener, LocationListener {
  public void onCreate(Bundle savedInstanceState) {
    LocationManager lm = (LocationManager) this.getSystemService(
   Context.LOCATION_SERVICE);
    if(lm.getAllProviders().contains(LocationManager.GPS_PROVIDER)){
      if(lm.isProviderEnabled(LocationManager.GPS_PROVIDER)){
        lm.addGpsStatusListener(this);
        // here the request was removed
        statusView.setText(?GPS service started?);
      } else {
        statusView.setText(?Please enable GPS?);
        saveLocationButton.setEnabled(false);
      }
    }
  }
  public void onPause() {
    lm.removeUpdates(this);
  }
  public void onResume(){
    // request was moved to the 'visible' onResume() method
    lm.requestLocationUpdates(LocationManager.GPS_PROVIDER, 1000, 0,
   this);
  }
}
```

REFERENCES

- http://www.researchgate.net/publication/235705377_Removing_Energy_Code_Smells_with_Reengineering_Services/file/79e41512c806145a21.pdf
- http://developer.android.com/guide/components/activities.html#ImplementingLifecycleCallbacks

Related Smells

- Bulk Data Transfer On Slow Network (4.1)
- Data Transmission Without Compression (4.2)
- Durable WakeLock (4.5)
- Prohibited Data Transfer (4.20)
- Rigid AlarmManager (4.22)

4.7 Inefficient Data Format And Parser

Affected Qualites
efficiency
Context
network, io
Tags
network, efficiency, io
Description
The use of Tree Parsers is slow. This chart is taken from: `http://www.google.com/events/io/2009/sessions/CodingLifeBatteryLife.html`. Grey bars represent Tree Parsers, whereas blue bars represent Event/Stream Parsers.

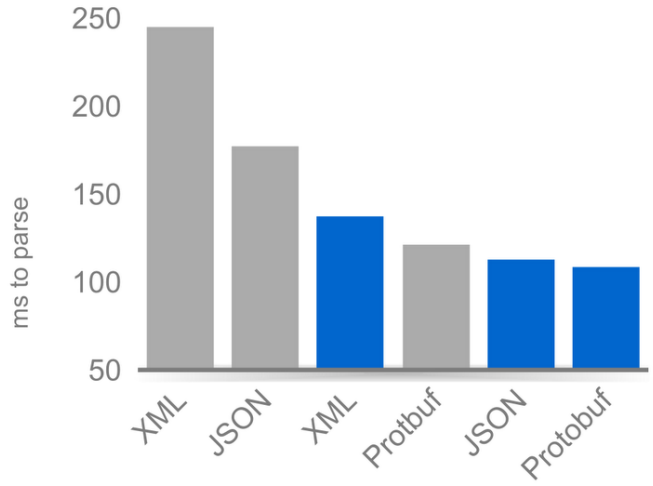


Figure 4.1: *Comparison of data formats and parsers.*

Refactorings

Use efficient data parser and format

Resolved Qualities
efficiency
Affected Qualities
-
Description
In general it is recommended to use stream parsers instead of tree parsers. It could be also beneficial to consider binary formats that can easily mix binary and text data into a single request

Note that according to an Android Issue the use of `android.util.Scanner` is very inefficient.

References

- https://groups.google.com/forum/#!msg/android-developers/7E-JgVAcjjk/E3wGv03wZVoJ
- https://code.google.com/p/android/issues/detail?id=57050
- http://www.google.com/events/io/2009/sessions/CodingLifeBatteryLife.html
- http://developer.android.com/reference/android/net/ConnectivityManager.html

RELATED SMELLS

- Bulk Data Transfer On Slow Network (4.1)
- Data Transmission Without Compression (4.2)
- Inefficient Data Structure (4.8)
- Inefficient SQL Query (4.9)
- Internal Getter/Setter (4.10)
- Member-Ignoring Method (4.14)
- Nested Layout (4.15)
- Network & IO Operations In Main Thread (4.16)
- Overdrawn Pixel (4.19)
- Prohibited Data Transfer (4.20)
- Slow Loop (4.24)
- Uncached Views (4.26)

4.8 Inefficient Data Structure

AFFECTED QUALITES
efficiency
CONTEXT
implementation
TAGS
data structure, efficiency
DESCRIPTION
The use of default data structures could be slow under certain circumstances, e.g. `HashMap <Integer, Object>`.

Refactorings

Use Efficient Data Structures

RESOLVED QUALITIES
efficiency
AFFECTED QUALITIES
-
DESCRIPTION
Instead of using `HashMap<Integer, Object>` the Android optmized `SparseArray` is recommended:

Code 4.8.1: Use of SparseArray

```
SparseArray<Bitmap> bitmaps = new SparseArray<Bitmap>()
bitmaps.append(i, newBitmap)
```

This is also transferable to:

- SparseLongArray
- SparseIntArray
- SparseBooleanArray
- LongSparseArray

REFERENCES

- *Pro Android Apps Performance Optimization* by [Gui12]
- https://speakerdeck.com/cyrilmottier/optimizing-android-ui-pro-tips-for-creating-smooth-and-responsive-apps
- http://www.programmingmobile.com/2012/07/android-performance-tweaking-parsearray.html
- http://developer.android.com/reference/android/util/SparseArray.html
- http://developer.android.com/reference/android/util/SparseLongArray.html

- http://developer.android.com/reference/android/util/SparseIntArray.html
- http://developer.android.com/reference/android/util/SparseBooleanArray.html
- http://developer.android.com/reference/android/util/LongSparseArray.html

RELATED SMELLS

- Inefficient Data Format And Parser (4.7)
- Inefficient SQL Query (4.9)
- Internal Getter/Setter (4.10)
- Member-Ignoring Method (4.14)
- Nested Layout (4.15)
- Overdrawn Pixel (4.19)
- Slow Loop (4.24)
- Uncached Views (4.26)

4.9 Inefficient SQL Query

Affected Qualites
efficiency
Context
implementation, network, database
Tags
network, database, efficiency
Description
To retrieve data from a database a SQL query is sent over *JDBC* connection to a remote server.

Refactorings

Use JSON query

Resolved Qualities
efficiency
Affected Qualities
-
Description
According to an answer in *Programmers Stackexchange* the use of a SQL query is discouraged as it introduces a lot of overhead. It should be preferred to send a query to webserver and revieve e.g. a *JSON* response. This response could be compressed efficiently.

Beyond that the projection of a query should be minimised:

Code 4.9.1: SQL Query with overhead

```
/* BAD */
select * from verybigtable
/* BETTER */
select id, singlecolumn from verybigtable
```

References

- http://programmers.stackexchange.com/questions/170463

Related Smells

- Bulk Data Transfer On Slow Network (4.1)
- Data Transmission Without Compression (4.2)
- Inefficient Data Format And Parser (4.7)
- Inefficient Data Structure (4.8)
- Internal Getter/Setter (4.10)
- Member-Ignoring Method (4.14)
- Nested Layout (4.15)

- Network & IO Operations In Main Thread (4.16)
- Prohibited Data Transfer (4.20)
- Overdrawn Pixel (4.19)
- Slow Loop (4.24)
- Uncached Views (4.26)

4.10 Internal Getter/Setter

AFFECTED QUALITES
efficiency
CONTEXT
implementation
TAGS
idiom, efficiency
DESCRIPTION
Internal fields are accessed from the same class via getters and setters. But according to `http://developer.android.com/training/articles/perf-tips.html#GettersSetters` Android virtual method are expensive.

Refactorings

Access Field Directly

RESOLVED QUALITIES
efficiency
AFFECTED QUALITIES
-
DESCRIPTION
Consider accessing the fields directly and only use getters and setters in public API.
Taken from the Performance Tips:

> Without a JIT, direct field access is about 3x faster than invoking a trivial getter. With the JIT (where direct field access is as cheap as accessing a local), direct field access is about 7x faster than invoking a trivial getter.

Note that inner class that utilizes private attributes of the parent class are compiled automatically to use getters and setters. The use of the `package` modifier for these attributes might be advisable.

REFERENCES

- `http://stackoverflow.com/questions/6716442/android-performance-avoid-internal-getters-setters`
- `http://developer.android.com/training/articles/perf-tips.html#GettersSetters`

RELATED SMELLS

- Inefficient Data Format And Parser (4.7)
- Inefficient Data Structure (4.8)
- Inefficient SQL Query (4.9)
- Member-Ignoring Method (4.14)
- Nested Layout (4.15)
- Overdrawn Pixel (4.19)

- Slow Loop (4.24)
- Uncached Views (4.26)
- Unclosed Closable (4.27)

4.11 Interrupting From Background

Affected Qualites
user conformity, user experience
Context
ui
Tags
user conformity, user experience
Description
It is a bad smell to start activities from *BroadcastRecievers* or *Services* that work in the background. Imagine a user writes an email and gets interrupted by another apps activity that just started. The worse: this annoying app might catche some of the input. This is also true for *Toast* messages that are sent by background apps.

Refactorings

Remove 'startActivity()' from background

Resolved Qualities
user conformity, user experience
Affected Qualities
-
Description
BroadcastRecievers and *Services* should not start another *Activity*, nor show a toast to inform the user that something happens. Use notifications instead.

References

- http://developer.android.com/guide/practices/seamlessness.html#interrupt
- http://developer.android.com/guide/topics/ui/notifiers/toasts.html
- http://developer.android.com/guide/topics/ui/notifiers/notifications.html

Related Smells

- Bulk Data Transfer On Slow Network (4.1)
- Dropped Data (4.4)
- Not Descriptive UI (4.18)
- Network & IO Operations In Main Thread (4.16)
- Prohibited Data Transfer (4.20)
- Public Data (4.21)
- Set Config Changes (4.23)
- Tracking Hardware Id (4.25)
- Uncontrolled Focus Order (4.28)

- Unnecessary Permission (4.29)
- Untouchable (4.30)

4.12 Leaking Inner Class

Affected Qualites
memory efficiency
Context
implementation
Tags
memory, class, leak, view
Description
Non-static inner classes holds a reference to the outer class. This could lead to a memory leak.

Here are two examples: One inner class and one anonymous inner class.

Code 4.12.1: Leaking class example

```
public class SampleActivity extends Activity {
  private final Handler mLeakyHandler = new Handler() {
    @Override
    public void handleMessage(Message msg) {
      // ...
    }
  }
  @Override
  protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    // Post a message and delay its execution for 10 minutes.
    mLeakyHandler.postDelayed(new Runnable() {
      @Override
      public void run() {
        // do anything
      }
    }, 60 * 10 * 1000);
    // Go back to the previous Activity.
    finish();
  }
}
```

Refactorings

Introduce Static Class

Resolved Qualities
memory efficiency
Affected Qualities
-
Description
Declare a static instance of the class:

Code 4.12.2: Leaking class example

```
/**
 * Instances of anonymous classes do not hold an implicit
 * reference to their outer class when they are "static".
 */
private static final Runnable sRunnable = new Runnable() {
  @Override
  public void run() {
    ...
  }
}
```

Introduce Weak Reference

RESOLVED QUALITIES
memory efficiency
AFFECTED QUALITIES
-
DESCRIPTION
Use a WeakReference. Objects hold by WeakReferences will still be garbage collected.

Code 4.12.3: Leaking class example

```
public class SampleActivity extends Activity {
  private static class MyHandler extends Handler {
    private final WeakReference<SampleActivity> mActivity;
    public MyHandler(SampleActivity activity) {
      mActivity = new WeakReference<SampleActivity>(activity);
    }
    @Override
    public void handleMessage(Message msg) {
      SampleActivity activity = mActivity.get();
      if (activity != null) {
        ...
      }
    }
  }
  private final MyHandler mHandler = new MyHandler(this);
  @Override
  protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    // Post a message and delay its execution for 10 minutes.
    mHandler.postDelayed(... , 600000);
    // Go back to the previous Activity.
    finish();
  }
}
```

References

- http://www.androiddesignpatterns.com/2013/01/inner-class-handler-memory-leak.html
- http://www.google.com/events/io/2011/sessions/memory-management-for-android-apps.html
- http://developer.android.com/reference/java/lang/ref/WeakReference.html

Related Smells

- Leaking Thread (4.13)
- Nested Layout (4.15)
- No Low Memory Resolver (4.17)
- Set Config Changes (4.23)
- Uncached Views (4.26)
- Unclosed Closable (4.27)

4.13 Leaking Thread

Affected Qualites
memory efficiency
Context
implementation
Tags
thread, memory, leak, thread, class
Description
In Java threads are GC roots, that is they are kept in the runtime and does not get collected. So if an *Activity* starts a thread and does not stop it this is considered a bug. As it can leak memory.

In this example the thread will never be stopped.

Code 4.13.1: Leaking thread

```
public class MainActivity extends Activity {

  private MyThread mThread;

  @Override
  protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    doIt();
  }
  private void doIt() {
    mThread = new MyThread();
    mThread.start();
  }
  private static class MyThread extends Thread {
    @Override
    public void run() {
      while (true) {
        doReallyHeavyStuff();
      }
    }
  }
}
```

Threads should only be used for short lived computations. The above mentioned behaviour also applies to AsyncTask.

Refactorings

Introduce Run Check Variable

Resolved Qualities
memory efficiency
Affected Qualities
-
Description
This will introduce a variable to check if the thread operation should still be performed.

Code 4.13.2: Introduce Run Check Variable

```
public class MainActivity extends Activity {

  private MyThread mThread;

  @Override
  protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    doIt();
  }
  private void doIt() {
    mThread = new MyThread();
    mThread.start();
  }
  private static class MyThread extends Thread {
    private boolean mRunning = false;
    @Override
    public void run() {
      mRunning = true;
      while (mRunning) {
        doReallyHeavyStuff();
      }
    }
    public void close() {
      mRunning = false;
    }
  }
  @Override
  protected void onDestroy() {
    super.onDestroy();
    mThread.close();
  }
}
```

Use fragments for configuration change

RESOLVED QUALITIES
memory efficiency
AFFECTED QUALITIES
-
DESCRIPTION
If there is a need to persist data across configuaration changes (due to orientation change, font size change) it is better to use a retained fragment.

This example is taken from the Android API samples:

Code 4.13.3: Use fragments for configuration change

```
/**
 * This example shows how you can use a Fragment to easily propagate
   state
 * (such as threads) across activity instances when an activity needs
   to be
 * restarted due to, for example, a configuration change.  This is a
   lot
 * easier than using the raw Activity.onRetainNonConfiguratinInstance
   () API.
 */
public class FragmentRetainInstance extends Activity {
  @Override
  protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    // First time init, create the UI.
    if (savedInstanceState == null) {
      getFragmentManager().beginTransaction().add(android.R.id.content
   ,
          new UiFragment()).commit();
    }
  }
  /**
   * This is a fragment showing UI that will be updated from work done
   * in the retained fragment.
   */
  public static class UiFragment extends Fragment {
    RetainedFragment mWorkFragment;
    @Override
    public View onCreateView(LayoutInflater inflater, ViewGroup
   container,
        Bundle savedInstanceState) {
      View v = inflater.inflate(R.layout.fragment_retain_instance,
   container, false);
      // Watch for button clicks.
      Button button = (Button)v.findViewById(R.id.restart);
      button.setOnClickListener(new OnClickListener() {
        public void onClick(View v) {
          mWorkFragment.restart();
        }
      });
      return v;
    }
    @Override
    public void onActivityCreated(Bundle savedInstanceState) {
      super.onActivityCreated(savedInstanceState);
      FragmentManager fm = getFragmentManager();
      // Check to see if we have retained the worker fragment.
      mWorkFragment = (RetainedFragment)fm.findFragmentByTag("work");
      // If not retained (or first time running), we need to create it
   .
```

```
    if (mWorkFragment == null) {
      mWorkFragment = new RetainedFragment();
      // Tell it who it is working with.
      mWorkFragment.setTargetFragment(this, 0);
      fm.beginTransaction().add(mWorkFragment, "work").commit();
    }
  }
}
/**
 * This is the Fragment implementation that will be retained across
 * activity instances.  It represents some ongoing work, here a
 thread
 * we have that sits around incrementing a progress indicator.
 */
public static class RetainedFragment extends Fragment {
  ProgressBar mProgressBar;
  int mPosition;
  boolean mReady = false;
  boolean mQuiting = false;
  /**
   * This is the thread that will do our work.  It sits in a loop
 running
   * the progress up until it has reached the top, then stops and
 waits.
   */
  final Thread mThread = new Thread() {
    @Override
    public void run() {
      // We'll figure the real value out later.
      int max = 10000;
      // This thread runs almost forever.
      while (true) {
        // Update our shared state with the UI.
        synchronized (this) {
          // Our thread is stopped if the UI is not ready
          // or it has completed its work.
          while (!mReady || mPosition >= max) {
            if (mQuiting) {
              return;
            }
            try {
              wait();
            } catch (InterruptedException e) {
            }
          }
          // Now update the progress.  Note it is important that
          // we touch the progress bar with the lock held, so it
          // doesn't disappear on us.
          mPosition++;
          max = mProgressBar.getMax();
          mProgressBar.setProgress(mPosition);
        }
```

```
        // Normally we would be doing some work, but put a kludge
        // here to pretend like we are.
        synchronized (this) {
          try {
            wait(50);
          } catch (InterruptedException e) {
          }
        }
      }
    }
  };
  /**
   * Fragment initialization.  We way we want to be retained and
   * start our thread.
   */
  @Override
  public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    // Tell the framework to try to keep this fragment around
    // during a configuration change.
    setRetainInstance(true);
    // Start up the worker thread.
    mThread.start();
  }
  /**
   * This is called when the Fragment's Activity is ready to go,
after
   * its content view has been installed; it is called both after
   * the initial fragment creation and after the fragment is re-
attached
   * to a new activity.
   */
  @Override
  public void onActivityCreated(Bundle savedInstanceState) {
    super.onActivityCreated(savedInstanceState);
    // Retrieve the progress bar from the target's view hierarchy.
    mProgressBar = (ProgressBar)getTargetFragment().getView().
findViewById(
        R.id.progress_horizontal);
    // We are ready for our thread to go.
    synchronized (mThread) {
      mReady = true;
      mThread.notify();
    }
  }
  /**
   * This is called when the fragment is going away.  It is NOT
called
   * when the fragment is being propagated between activity
instances.
   */
  @Override
```

```
    public void onDestroy() {
      // Make the thread go away.
      synchronized (mThread) {
        mReady = false;
        mQuiting = true;
        mThread.notify();
      }
      super.onDestroy();
    }
    /**
     * This is called right before the fragment is detached from its
     * current activity instance.
     */
    @Override
    public void onDetach() {
      // This fragment is being detached from its activity.  We need
      // to make sure its thread is not going to touch any activity
      // state after returning from this function.
      synchronized (mThread) {
        mProgressBar = null;
        mReady = false;
        mThread.notify();
      }
      super.onDetach();
    }
    /**
     * API for our UI to restart the progress thread.
     */
    public void restart() {
      synchronized (mThread) {
        mPosition = 0;
        mThread.notify();
      }
    }
  }
}
```

REFERENCES

- http://www.androiddesignpatterns.com/2013/04/activitys-threads-memory-leaks.html
- http://www.androiddesignpatterns.com/2013/04/retaining-objects-across-config-changes.html
- http://developer.android.com/guide/topics/resources/runtime-changes.html
- https://android.googlesource.com/platform/development/+/master/samples/ApiDemos/src/com/example/android/apis/app/FragmentRetainInstance.java

- http://developer.android.com/training/displaying-bitmaps/cache-bitmap.html#config-changes

RELATED SMELLS

- Leaking Inner Class (4.12)
- Nested Layout (4.15)
- No Low Memory Resolver (4.17)
- Set Config Changes (4.23)
- Uncached Views (4.26)
- Unclosed Closable (4.27)

4.14 Member-Ignoring Method

Affected Qualites
efficiency
Context
implementation
Tags
idiom, efficiency
Description
A class has non-static methods that do not access any property.

Code 4.14.1: Method without field access

```
public class MyClass {
  private String myProp;

  public int returnZero() {
    return 0;
  }
}
```

Refactorings

Introduce Static Method

Resolved Qualities
efficiencyties
Affected Qualities
-
Description
Make the method static.

Code 4.14.2: With static method

```
public class MyClass {
  private String myProp;

  public static int returnZero() {
    return 0;
  }
}
```

References

- http://developer.android.com/training/articles/perf-tips.html#PreferStatic

Related Smells

- Inefficient Data Format And Parser (4.7)
- Inefficient Data Structure (4.8)
- Inefficient SQL Query (4.9)
- Internal Getter/Setter (4.10)
- Nested Layout (4.15)
- Overdrawn Pixel (4.19)
- Slow Loop (4.24)
- Unclosed Closable (4.27)
- Uncached Views (4.26)

4.15 Nested Layout

Affected Qualites
efficiency, user experience, startup time
Context
ui
Tags
layout, view, efficiency
Description
Layouts with elements that have the attribute `weight` set must be computed twice. While each new element requires initialisation, layout and drawing parsing deep nested *LinearLayouts* will also increase the computation time exponentially.

Refactorings

Flatten Layouts

Resolved Qualities
efficiency, user experience, startup time
Affected Qualities
-
Description
Nested *LinearLayouts* could be flattened by the use of *RelativeLayouts*. Or by the use of the `<include>`-Tag. The tools *hierarchyviewer* and *layoutopt* identify too complex layouts and proposes optimizations.

References

- *Pro Android Apps Performance Optimization* by [Gui12]
- `http://developer.android.com/tools/help/hierarchy-viewer.html`
- `http://developer.android.com/tools/help/layoutopt.html`
- `http://developer.android.com/training/improving-layouts/optimizing-layout.html`

Related Smells

- Inefficient Data Format And Parser (4.7)
- Inefficient Data Structure (4.8)
- Inefficient SQL Query (4.9)
- Internal Getter/Setter (4.10)
- Leaking Inner Class (4.12)
- Leaking Thread (4.13)
- Member-Ignoring Method (4.14)
- Overdrawn Pixel (4.19)
- Set Config Changes (4.23)
- Slow Loop (4.24)

- Uncached Views (4.26)

4.16 Network & IO Operations In Main Thread

AFFECTED QUALITES
user experience
CONTEXT
ui, implementation, io, network
TAGS
user experience, network, io
DESCRIPTION
The main thread is where the *UI* lives in. Therefor it is not recommended to do heavy operations (network, io, sql) in it.

Refactorings

Use StrictMode

RESOLVED QUALITIES
user experience
AFFECTED QUALITIES
-
DESCRIPTION
From Android Developer Documentation:

> StrictMode is a developer tool which detects things you might be doing by accident and brings them to your attention so you can fix them. It is policy on a thread that lets you set what is not allowed and how you will be noticed.

So *StrictMode* helps to find code smells. It can be enabled in *Application*, *Activity* or in another applications `onCreate()` method. See this very basic example:

Code 4.16.1: Test

```
public void onCreate() {
  if (DEVELOPER_MODE) {
    StrictMode.setThreadPolicy(new StrictMode.ThreadPolicy.Builder()
      .detectDiskReads()
      .detectDiskWrites()
      .detectNetwork()   // or .detectAll() for all detectable
   problems
      .penaltyLog()
      .build());
    StrictMode.setVmPolicy(new StrictMode.VmPolicy.Builder()
      .detectLeakedSqlLiteObjects()
      .detectLeakedClosableObjects()
      .penaltyLog()
      .penaltyDeath()
      .build());
  }
  super.onCreate();
}
```

Be careful to **do not publish applications** with StrictMode turned on.

References

- http://android-developers.blogspot.de/2010/12/new-gingerbread-api-strictmode.html
- http://developer.android.com/reference/android/os/StrictMode.html

Related Smells

- Bulk Data Transfer On Slow Network (4.1)
- Data Transmission Without Compression (4.2)
- Inefficient Data Format And Parser (4.7)
- Interrupting From Background (4.11)
- Inefficient SQL Query (4.9)
- Not Descriptive UI (4.18)
- Prohibited Data Transfer (4.20)
- Uncontrolled Focus Order (4.28)
- Untouchable (4.30)

4.17 No Low Memory Resolver

AFFECTED QUALITES
memory efficiency, stability, user experience
CONTEXT
implementation
TAGS
memory
DESCRIPTION
Mobile systems usually have little *R*AM and no *SWAP* space to free some memory. Android provides a mechanism to help the system manage memory. The overrideable method `Activity.onLowMemory()` is called when all background process have been killed. That is, before reaching the point of killing processes hosting service and foreground UI that we would like to avoid killing.

This method should clean caches or unnecessary resources. If this method is not implemented this is considered a smell, due to the fact that it can cause abnormal programm termination.

Refactorings

Override onLowMemory()

RESOLVED QUALITIES
memory efficiency, stability, user experience
AFFECTED QUALITIES
-
DESCRIPTION
Beschreibung

Code 4.17.1: Override onLowMemory(

```
class MyActivity extends Activity {

  ...

  @Override
  public void onLowMemory() {
    // clean caches and unnecessary ressources
  }
}
```

REFERENCES

- *Pro Android Apps Performance Optimization* by [Gui12]
- `http://developer.android.com/reference/android/app/Activity.html`

RELATED SMELLS

- Leaking Inner Class (4.12)

- Leaking Thread (4.13)
- Set Config Changes (4.23)
- Unclosed Closable (4.27)

4.18 Not Descriptive UI

AFFECTED QUALITES
accessibility, user conformity, user experience
CONTEXT
ui
TAGS
accessibility, user conformity, user experience
DESCRIPTION
For visually or otherwise physically impaired people it is hard to navigate the app and keep track what is shown. To help these people Android porived TalkBack, an app that reads the contents out.

So every UI element should have content description what it is for. Several elements as labels, and text fields do have this intentionally, but others have not:

- ImageButton
- ImageView
- EditText

If these elements have not set `android:contentDescription` in the XML (or via `setContentDescription()`), then it is considered a smell.

Refactorings

Set content description

RESOLVED QUALITIES
accessibility, user conformity, user experienc
AFFECTED QUALITIES
-
DESCRIPTION
Use the XML attribute `android:contentDescription` or set it by calling `element.setContentDescription()`.

REFERENCES

- `http://developer.android.com/guide/topics/ui/accessibility/apps.html#label-ui`
- `http://android-developers.blogspot.de/2012/04/accessibility-are-you-serving-all-your.html`

RELATED SMELLS

- Bulk Data Transfer On Slow Network (4.1)
- Dropped Data (4.4)
- Interrupting From Background (4.11)
- Network & IO Operations In Main Thread (4.16)
- Prohibited Data Transfer (4.20)
- Public Data (4.21)

- Set Config Changes (4.23)
- Tracking Hardware Id (4.25)
- Uncontrolled Focus Order (4.28)
- Unnecessary Permission (4.29)
- Untouchable (4.30)

4.19 Overdrawn Pixel

AFFECTED QUALITES
efficiency
CONTEXT
ui
TAGS
efficiency
DESCRIPTION
The layout of an Android UI is generally build using XML. So you can stack several layers to form complex user interfaces. Containers and basic elements may have attributes to describe their background, text, border, etc. In this case it might be possible to overdraw a pixel. This means for example the root container has a black background and contains several other elements that havve another (non-transparent) background color. And on top of these are buttons, text-fields that are also colored. This means that a pixel must be drawn several times which is a costly process.

Figure 4.2 shows overdrawn pixels from best to worst (blue, green, light red, red). It can be enabled as of Android 4.1 in the „Show Overdraw GPU“ developer settings.

Figure 4.2: *Overdrawn Pixel*

Refactorings

Flatten Layouts

Resolved Qualities
efficiency, user experience, startup time
Affected Qualities
-
Description
Nested *LinearLayouts* could be flattened by the use of *RelativeLayouts*. Or by the use of the `<include>`-Tag. The tools *hierarchyviewer* and *layoutopt* identify too complex layouts and proposes optimizations.

References

- `https://developers.google.com/events/io/sessions/325418001`
- `http://www.curious-creature.org/docs/android-performance-case-study-1.html`

Related Smells

- Inefficient Data Format And Parser (4.7)
- Inefficient Data Structure (4.8)
- Internal Getter/Setter (4.10)
- Member-Ignoring Method (4.14)
- Nested Layout (4.15)
- Slow Loop (4.24)
- Uncached Views (4.26)
- Inefficient SQL Query (4.9)

4.20 Prohibited Data Transfer

AFFECTED QUALITES
energy efficiency, user conformity, user experience
CONTEXT
network
TAGS
energy, user conformity, user experience, network
DESCRIPTION
Due to possible traffic restrictions in current mobile charges the user expects that no unnecessary or large data is transmitted without permission.

Refactorings

Check background data transfer

RESOLVED QUALITIES
energy efficiency, user conformity, user experience
AFFECTED QUALITIES
-
DESCRIPTION
It should be checked if the user has disabled background data transmission before transmitting.

Code 4.20.1: Check for enabled background data transmission

```
ConnectivityManager mConnectivity;
TelephonyManager mTelephony;
// Skip if no connection, or background data disabled
NetworkInfo info = mConnectivity.getActiveNetworkInfo();
if (info == null || !mConnectivity.getBackgroundDataSetting()) {
  return false;
}
```

Note, from Android Developer Documentation:

> This method was deprecated in API level 14. As of ICE_CREAM_SANDWICH, availability of background data depends on several combined factors, and this method will always return true. Instead, when background data is unavailable, getActiveNetworkInfo() will now appear disconnected.

Another approach will be to provide a user setting that controls background data transmission under certain circumstances (WiFi, Edge, 3G, 4G).

REFERENCES

- http://www.google.com/events/io/2009/sessions/CodingLifeBatteryLife.html
- http://developer.android.com/reference/android/net/ConnectivityManager.html#getBackgroundDataSetting()

Related Smells

- Bulk Data Transfer On Slow Network (4.1)
- Data Transmission Without Compression (4.2)
- Dropped Data (4.4)
- Durable WakeLock (4.5)
- Early Resource Binding (4.6)
- Inefficient Data Format And Parser (4.7)
- Inefficient SQL Query (4.9)
- Interrupting From Background (4.11)
- Network & IO Operations In Main Thread (4.16)
- Not Descriptive UI (4.18)
- Public Data (4.21)
- Rigid AlarmManager (4.22)
- Set Config Changes (4.23)
- Tracking Hardware Id (4.25)
- Unnecessary Permission (4.29)

4.21 Public Data

Affected Qualites
security, user conformity, user experience
Context
implementation
Tags
security, user conformity, user experience
Description
Private data is kept in a store that is publicly accessible (by other applications). Reading/writing data is usally done by:

Code 4.21.1: Security settings while opening a file

```
Context.openFileOutput(String name, int mode)
Context.getSharedPreferences(String name, int mode)
Context.openOrCreateDatabase(String name, int mode, SQLiteDatabase.
    CursorFactory factory)
```

whereas `int mode` will refer to:

- `Context.MODE_PRIVATE`
- `Context.MODE_WORLD_READABLE` → Might indicate not-intended public data
- `Context.MODE_WORLD_WRITEABLE` → Might indicate not-intended public data
- `Context.MODE_MULTI_PROCESS`

Refactorings

Set private mode

Resolved Qualities
security, user conformity, user experience
Affected Qualities
-
Description
For private data use the flag `Context.MODE_PRIVATE`.

References

- https://developers.google.com/events/io/2012/sessions/gooio2012/107/
- http://developer.android.com/reference/android/content/Context.html#MODE_PRIVATE

Related Smells

- Bulk Data Transfer On Slow Network (4.1)
- Debuggable Release (4.3)
- Dropped Data (4.4)

- Prohibited Data Transfer (4.20)
- Interrupting From Background (4.11)
- Not Descriptive UI (4.18)
- Set Config Changes (4.23)
- Tracking Hardware Id (4.25)
- Unnecessary Permission (4.29)

4.22 Rigid AlarmManager

AFFECTED QUALITES
efficiency, energy consumption
CONTEXT
implementation
TAGS
energy, alarm, efficiency
DESCRIPTION
With the use of AlarmManager it is possible that operations can be executed at a sepcific moment in the future. It is possible that several operations got executed. Every AlarmManager-triggered operation wakes up the phone, so the overall use of energy and CPU might be higher, then if bundled together.

Code 4.22.1: AlarmManager example

```
AlarmManager am = (AlarmManager) context.getSystemService(Context.
    ALARM_SERVICE);
Intent intent = new Intent(context, MyService.class);
PendingIntent pendingIntent = PendingIntent.getService(context, 0,
    intent, 0);
long interval = DateUtils.MINUTE_IN_MILLIS * 30;
long firstWake = System.currentTimeMillis() + interval;
am.setRepeating(AlarmManager.RTC_WAKEUP, firstWake, interval,
    pendingIntent);
```

Refactorings

Set Inexact Alarmmanager

RESOLVED QUALITIES
efficiency, energy consumption
AFFECTED QUALITIES
-
DESCRIPTION
To ensure that the system is able to bundle several updates together, it is recommended to use:

Code 4.22.2: AlarmManager example with inexact repeating

```
AlarmManager.setInexactRepeating(int type, long triggerAtMillis, long
    intervalMillis, PendingIntent operation)
```

REFERENCES

- http://www.google.com/events/io/2009/sessions/CodingLifeBatteryLife.html

- http://developer.android.com/reference/android/app/AlarmManager.html

Related Smells

- Bulk Data Transfer On Slow Network (4.1)
- Data Transmission Without Compression (4.2)
- Durable WakeLock (4.5)
- Early Resource Binding (4.6)
- Prohibited Data Transfer (4.20)

4.23 Set Config Changes

Affected Qualites
user conformity, user experience
Context
implementation
Tags
memory, leak, user conformity, user experience, view
Description
It is considered a smell to set the attribute android:configChanges in the Android manifest. This attribute defines what configuration changes the app has to handle manually (else the system will take care of persisting input data between config changes in the activities ui elements). Handling them manually can cause memory bugs (leaving resources in memory).

Refactorings

Use fragments for configuration change

Resolved Qualities
memory efficiency, user conformity, user experience
Affected Qualities
-
Description
If there is a need to persist data across configuaration changes (due to orientation change, font size change) it is better to use a retained fragment.

For an example see Code 4.13.3, which is taken from the Android API samples.

References

- http://www.androiddesignpatterns.com/2013/04/retaining-objects-across-config-changes.html
- http://developer.android.com/guide/topics/resources/runtime-changes.html
- https://android.googlesource.com/platform/development/+/master/samples/ApiDemos/src/com/example/android/apis/app/FragmentRetainInstance.java
- http://developer.android.com/training/displaying-bitmaps/cache-bitmap.html#config-changes

Related Smells

- Bulk Data Transfer On Slow Network (4.1)
- Dropped Data (4.4)
- Interrupting From Background (4.11)
- Leaking Inner Class (4.12)
- Leaking Thread (4.13)

- Nested Layout (4.15)
- No Low Memory Resolver (4.17)
- Not Descriptive UI (4.18)
- Public Data (4.21)
- Prohibited Data Transfer (4.20)
- Tracking Hardware Id (4.25)
- Uncached Views (4.26)
- Unclosed Closable (4.27)
- Unnecessary Permission (4.29)

4.24 Slow Loop

Affected Qualites
efficiency
Context
implementation
Tags
idiom, efficiency
Description
A slow version of a for-loop is used.

Refactorings

Enhance For-Loop

Resolved Qualities
efficiency
Affected Qualities
-
Description
Consider the following code taken from these performance tips: http://developer.android.com/training/articles/perf-tips.html#Loops

Code 4.24.1: Enhanced for-loop

```
static class Foo {
  int mSplat;
}

Foo[] mArray = ...

public void zero() {
  int sum = 0;
  for (int i = 0; i < mArray.length; ++i) {
    sum += mArray[i].mSplat;
  }
}
public void one() {
  int sum = 0;
  Foo[] localArray = mArray;
  int len = localArray.length;

  for (int i = 0; i < len; ++i) {
    sum += localArray[i].mSplat;
  }
}
public void two() {
  int sum = 0;
  for (Foo a : mArray) {
    sum += a.mSplat;
  }
}
```

`two()` is the fastest loop for devices without *JIT* and indistinguishable from `one()` for devices with *JIT*. But consider a hand-written counted loop for performance-critical *ArrayList* iteration. The *JIT* was introduced in Android Version 2.2 (Froyo).

References

- `http://developer.android.com/training/articles/perf-tips.html#Loops`

Related Smells

- Inefficient Data Format And Parser (4.7)
- Inefficient Data Structure (4.8)
- Internal Getter/Setter (4.10)
- Member-Ignoring Method (4.14)
- Nested Layout (4.15)
- Overdrawn Pixel (4.19)
- Uncached Views (4.26)
- Unclosed Closable (4.27)
- Inefficient SQL Query (4.9)

4.25 Tracking Hardware Id

Affected Qualites
security, user conformity, user experience
Context
implementation
Tags
security, user conformity, user experience
Description
For some use cases it might be necessary to get a unique, reliable, unique device identifier.

This could be achieved by reading the IMEI, MEID, or ESN of the phone by calling `TelephonyManager.getDeviceId()`. But this needs the permission `READ_PHONE_STATE`, which might be considered unnecessary/harmful for the user. Besides the fact that another supsicious permission that can track the user-phone combination is required, it is not stable nor reliable.

Unique „Usually unique“ See `http://en.wikipedia.org/wiki/International_Mobile_Station_Equipment_Identity`

Reliable On phones it is ok, but lacks on WiFi-only or music players devices

Stable According to „Identifying App Installations“ (see references) some devices return garbage.

Refactorings

Use unique generated Id

Resolved Qualities
security, user conformity, user experience
Affected Qualities
-
Description
The goal of the problem is to identify one concrete installation instead of a concrete hardware. `Settings.Secure.ANDROID_ID` is a 64-bit hex string that is generated when the device is booted the first time. It is reset when the device is wiped. The downside of this approach is the fact that it is not reliable in Android Versions before Froyo (2.2). In „Identifying App Installations“ (see references) it is said there is at least one known manufacturer bug that returns the same id on every device. To target also older versions a legacy fallback method could be implemented that generates and stores a unique id at the first run of the app, eg. with the help of:

Code 4.25.1: Generate random Id

```
UUID.randomUUID().toString();
```

References

- http://android-developers.blogspot.de/2011/03/identifying-app-installations.html
- http://developer.android.com/reference/android/telephony/TelephonyManager.html#getDeviceId()
- http://developer.android.com/reference/android/Manifest.permission.html#READ_PHONE_STATE
- http://developer.android.com/reference/android/provider/Settings.Secure.html#ANDROID_ID
- http://developer.android.com/reference/java/util/UUID.html

Related Smells

- Bulk Data Transfer On Slow Network (4.1)
- Debuggable Release (4.3)
- Dropped Data (4.4)
- Interrupting From Background (4.11)
- Not Descriptive UI (4.18)
- Prohibited Data Transfer (4.20)
- Public Data (4.21)
- Set Config Changes (4.23)
- Unnecessary Permission (4.29)

4.26 Uncached Views

Affected Qualites
efficiency
Context
ui
Tags
view, efficiency
Description
Scrolling of *ListViews* or switching between pages of *ViewPager* could be slow. Code 4.26.1 contains a lot of expensive `findViewById()` calls which are frequently called while scrolling.

Code 4.26.1: Wrong approach creating view

```
@Override
public View getView(int position, View convertView, ViewGroup parent)
    {
  LayoutInflater li = (LayoutInflater) getContext().getSystemService(
    Context.LAYOUT_INFLATER_SERVICE);
  final View view = li.inflate(R.layout.objectData, parent, false);
  ((TextView) view.findViewById(android.R.id.text1)).setText("FOO")
  return view;
}
```

This will inflate a *View* every time a new *View* is rendered (scrolled to).

Refactorings

Introduce View Holder

Resolved Qualities
efficiency
Affected Qualities
-
Description
To make *ListViews* or any view holding views (like *ViewPager*) more smooth, you have to create a class *ViewHolder* that holds all fields of the view:

Code 4.26.2: ViewHolder class

```
static class ViewHolder {
  TextView text;
  TextView timestamp;
  ImageView icon;
  ProgressBar progress;
  int position;
}
```

Then the *Adapter* is fixed:

Code 4.26.3: Example for the user of ViewHolder

```
class MyAdapter extends ArrayAdapter {
  ...
  @Override
  public View getView(int position, View v, ViewGroup parent) {
  ViewHolder viewHolder;
    if (v == null) {
      LayoutInflater li = (LayoutInflater) getContext().
   getSystemService(
        Context.LAYOUT_INFLATER_SERVICE);
      v = li.inflate(R.layout.objectData, parent, false);
      viewHolder = new ViewHolder();
      viewHolder.txText = (TextView) v.findViewById(R.id.vText);
      viewHolder.txTimestamp = (TextView) v.findViewById(R.id.
   vTimestamp);
      viewHolder.txIcon = (TextView) v.findViewById(R.id.vIcon);
      viewHolder.txProgress = (TextView) v.findViewById(R.id.vProgress
   );
      viewHolder.txPosition = (TextView) v.findViewById(R.id.vPosition
   );
      v.setTag(viewHolder);
    } else {
      viewHolder = (ViewHolder) v.getTag();
    }
    // access fields and populate with data
    viewHolder.txtText(...)
    return v;
  }
}
```

Use convertView

Resolved Qualities
efficiency
Affected Qualities
-
Description
The use of the already inflated `convertView` parameter is faster:

Code 4.26.4: Example for convertView

```
@Override
public View getView(int position, View convertView, ViewGroup parent)
    {
    LayoutInflater li = (LayoutInflater) getContext().getSystemService
    (
            Context.LAYOUT_INFLATER_SERVICE);
    if(convertView == null) {
        convertView = li.inflate(R.layout.objectData, parent, false);
```

```
    }
    ((TextView) convertView.findViewById(android.R.id.text1)).setText(
   "FOO")
    return convertView;
}
```

References

- http://www.youtube.com/watch?v=wDBM6wVEO70
- http://developer.android.com/training/improving-layouts/smooth-scrolling.html

Related Smells

- Inefficient Data Format And Parser (4.7)
- Inefficient Data Structure (4.8)
- Internal Getter/Setter (4.10)
- Leaking Inner Class (4.12)
- Leaking Thread (4.13)
- Member-Ignoring Method (4.14)
- Nested Layout (4.15)
- Overdrawn Pixel (4.19)
- Set Config Changes (4.23)
- Slow Loop (4.24)
- Inefficient SQL Query (4.9)

4.27 Unclosed Closable

Affected Qualites
memory efficiency
Context
implementation
Tags
memory, leak, idiom
Description
An object implementing the `Closable` interface is not closed.

Refactorings

Close Closable

Resolved Qualities
memory efficiency
Affected Qualities
-
Description
The object should be closed properly with

Code 4.27.1: Close Closable

```
import java.io.Closable
Closable object = ...
// do anything with object
object.close();
```

Currently Android only supports Java 6. But if it will support Java 7 it is possible to use the `java.lang.AutoCloseable` interface in conjunction with the try-with-ressources statement:

Code 4.27.2: Close Closable

```
try (AutoCloseable object = ...) {
  object.doStuff();
}
```

References

- *Pro Android Apps Performance Optimization* by [Gui12]
- `http://developer.android.com/reference/java/io/Closeable.html`

Related Smells

- Internal Getter/Setter (4.10)
- Leaking Inner Class (4.12)

- Leaking Thread (4.13)
- Member-Ignoring Method (4.14)
- No Low Memory Resolver (4.17)
- Set Config Changes (4.23)
- Slow Loop (4.24)

4.28 Uncontrolled Focus Order

Affected Qualites
user experience, accessibility
Context
ui
Tags
accessibility, user experience, view
Description
Using directional controls means that there a four directions of navigation up, down, left (previous) and right (next). By default the Android system computes the nearest neighbor ui element and set it appropriatly. In some cases this might not be the desired (unlogical) effect.

Refactorings

Set focus order

Resolved Qualities
user experience, accessibility
Affected Qualities
-
Description
Use the XML attributes on *View* elements:

- `android:nextFocusDown`
- `android:nextFocusLeft`
- `android:nextFocusRight`
- `android:nextFocusUp`

It is also possible to set it at runtime via:

- `View.setNextFocusDownId()`
- `View.setNextFocusLeftId()`
- `View.setNextFocusRightId()`
- `View.setNextFocusUpId()`

It might be necessary to set the `android:focusable` attribute first (on elements that do not set it true by default).

References

- `http://developer.android.com/guide/topics/ui/accessibility/apps.html#focus-order`
- `http://android-developers.blogspot.de/2012/04/accessibility-are-you-serving-all-your.html`

Related Smells

- Interrupting From Background (4.11)

- Network & IO Operations In Main Thread (4.16)
- Not Descriptive UI (4.18)
- Untouchable (4.30)

4.29 Unnecessary Permission

AFFECTED QUALITES
security, user conformity, user experience
CONTEXT
implementation
TAGS
security, permission, user conformity, user experience
DESCRIPTION
The app requests several permissions, some of them are not needed, as they can be replaced easily.

The more permissions an app needs, the more suspicious it is for the user.

Refactorings

Use activity intent

RESOLVED QUALITIES
security, user conformity, user experience
AFFECTED QUALITIES
-
DESCRIPTION
From S*ecurity and Privacy in Android Apps* (see references):

> Permissions aren't required if you launch an activity that has the permission Security and Privacy in Android Apps

That is:

- getting a picture from the camera
- sending a short message from the messaging app
- selecting a contact

The user decides wether it executes this action and by what concrete app.

Code 4.29.1: Take camera picture

```
// create Intent to take a picture and return control to the calling
   application
Intent intent = new Intent(MediaStore.ACTION_IMAGE_CAPTURE);
// create a file to save the image
fileUri = getOutputMediaFileUri(MEDIA_TYPE_IMAGE);
// set the image file name
intent.putExtra(MediaStore.EXTRA_OUTPUT, fileUri);
// start the image capture Intent
startActivityForResult(intent, MY_REQUEST_CODE);
```

Code 4.29.2: Write short message

```
Uri smsNumber = Uri.parse("sms:5551212");
Intent intent = new Intent(Intent.ACTION_VIEW);
intent.setData(smsNumber);
intent.putExtra(Intent.EXTRA_TEXT, "hey there!");
startActivity(intent);
```

Code 4.29.3: Read a contact

```
public class MyActivity extends Activity {
  ...
  static final int PICK_CONTACT_REQUEST = 0;

  protected boolean onKeyDown(int keyCode, KeyEvent event) {
    if (keyCode == KeyEvent.KEYCODE_DPAD_CENTER) {
      // When the user presses center button, let them pick a contact.
      startActivityForResult(
        new Intent(Intent.ACTION_PICK,
        new Uri("content://contacts")),
        PICK_CONTACT_REQUEST);
      return true;
    }
    return false;
  }

  protected void onActivityResult(int requestCode, int resultCode,
    Intent data) {
    if (requestCode == PICK_CONTACT_REQUEST) {
      if (resultCode == RESULT_OK) {
        // A contact was picked.  Here we will just display it
        // to the user.
        startActivity(new Intent(Intent.ACTION_VIEW, data));
      }
    }
  }
}
```

References

- https://developers.google.com/events/io/2012/sessions/gooio2012/107/
- http://developer.android.com/guide/topics/media/camera.html#manifest
- http://developer.android.com/reference/android/app/Activity.html#StartingActivities

Related Smells

- Bulk Data Transfer On Slow Network (4.1)
- Debuggable Release (4.3)
- Dropped Data (4.4)
- Interrupting From Background (4.11)
- Not Descriptive UI (4.18)
- Prohibited Data Transfer (4.20)
- Public Data (4.21)
- Set Config Changes (4.23)
- Tracking Hardware Id (4.25)

4.30 Untouchable

Affected Qualites
user experience, accessibility
Context
ui
Tags
accessibility, user experience
Description
If a touchable ui elements size is less then 48dp (ca. 9mm) it is not easily touchable.

Refactorings

Increase touchable size

Resolved Qualities
user experience, accessibility
Affected Qualities
-
Description
Make the size of the touchable view element bigger.

References

- http://android-developers.blogspot.de/2012/04/accessibility-are-you-serving-all-your.html

Related Smells

- Interrupting From Background (4.11)
- Network & IO Operations In Main Thread (4.16)
- Not Descriptive UI (4.18)
- Uncontrolled Focus Order (4.28)

5 Realisierung mit Werkzeugunterstützung

Um den Ansatz der qualitätsorientierten Softwareentwicklung umzusetzen, ist es notwendig, eine Werkzeugunterstützung zur Verfügung zu stellen, welche *Quality Smells* schon in frühen Phasen automatisiert erkennt und refaktoriert. *Quality Smells* sind in unterschiedlicher Ausprägung plattformbezogen. Sie beziehen sich auf konkrete Hardware, eine Programmiersprache, eine Plattform wie Android oder auch nur eine bestimme Version einer Bibliothek. Aus diesem Grund sollte ein generischer Ansatz benutzt werden, der es ermöglicht *Quality Smells* für die unterschiedlichsten Plattformen zu beschreiben.

5.1 Einführung in Refactory

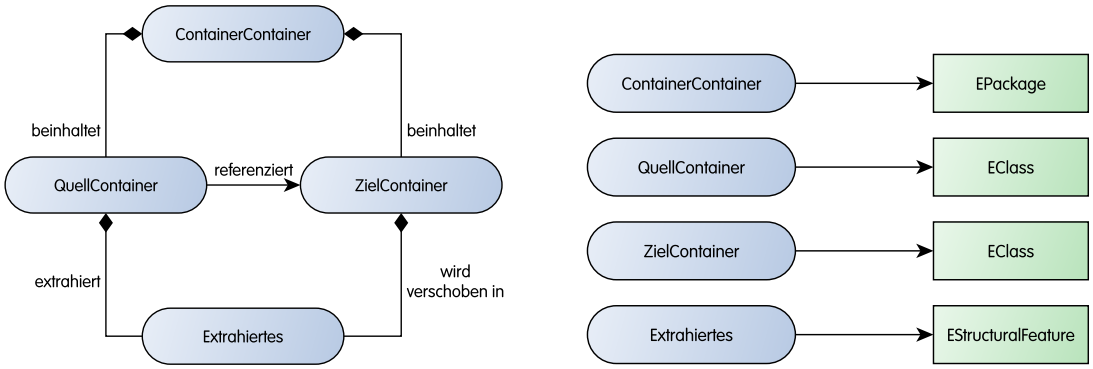
Die modellgetriebene Softwareentwicklung (MDSD) unterstützt die abstrakte Problembeschreibungen losgelöst von der plattformspezifischen Implementierung zu betrachten. Dafür haben sich domänenspezifische Sprachen und graphische Darstellungen zur Modellbeschreibung etabliert (UML), welche man dann zum Beispiel in Code transformieren kann. Die Vorteile der MDSD liegen in dem hohen Grad der Abstraktion der Domänenbeschreibung und damit in der besseren Kommunikation und in der hohen Wiederverwendbarkeit.

Reimann [Rei10] entwickelte deswegen aufbauend auf MDSD das Eclipse-basierte Werkzeug *Refactory*[1]. Ziel des Werkzeugs ist es, *Refactorings* generisch zu spezifizieren und diese in konkreten Modell auszuführen. Solche generischen Transformationsschritte sind beispielsweise „Extrahiere X“ oder „Benenne X um“. Dafür operiert es auf der M3-Metametamodellebene und verwendet das Eclipse Modelling Framework (EMF). Um die generischen Operationen auf konkreten Modellen auszuführen, wird ein rollenbasierter Ansatz von Riehle und Gross [RG98] genutzt. Im Allgemeinen beschreiben Rollen dabei das Verhalten und Kollaborationen von Elementen in einem System. In *Refactory* definiert ein Rollenmodell, welches Verhalten teilnehmende Elemente innerhalb eines abstrakten *Refactoring* einnehmen. Ein Beispiel für „Extrahiere X“ ist in Abbildung 5.1a zu sehen. In Abbildung 5.1b wird ein *Role Mapping* angegeben, dass den Rollen des generischen *Refactorings* Elemente des Zielmetamodells zuordnet. Als Zielmodell eignen sich alle EMF-kompatiblen Modelle, zum Beispiel das Java Metamodell *JaMoPP*[2] oder UML Zustandsautomaten.

Von Reimann und Aßmann [RA13] wurde das Werkzeug um das Erkennen und Auflösen von *Quality Smells* erweitert. Um die separate Weiterentwicklung und Wiederverwendbarkeit zu ermöglichen besteht das Framework aus drei Komponenten:

Quality Smell Repository Beinhaltet einen Katalog von *Quality Smells*. Sie beziehen sich auf ein konkretes Metamodell. Die Berechnung, welchen Einfluss ein *Smell* auf Qua-

[1] http://www.modelrefactoring.org
[2] http://www.jamopp.org



(a) *Rollenmodell* **(b)** Role Mapping *auf* ECore-*Modell*

Abbildung 5.1: *Rollenmodell und* Role Mapping *für das* Refactoring *„Extrahiere X", nach [RSA12]*

litäten hat, wird von *QualityCalculation* übernommen. Zudem ist es möglich Refactorings und Postprozessoren zum Auflösen von *Quality Smells* anzubieten.

Quality Smell Calculation Repository Beinhaltet Algorithmen zur Berechnung des Einflusses auf Qualitäten. Es lassen sich metrikbasierte oder strukturbasierte Berechnungsvorschriften angeben. Mit *IncQuery*[3] ist es möglich strukturbasierte Abfragen für *JaMoPP* zu definieren. Metrikbasierte Vorschriften lassen sich mit der General Purpose Language Java definieren.

Generic Model Refactoring Beinhaltet rollenbasierte, generische Modelltransformationsschritte. Wie oben beschrieben, lässt sich über ein konkretes *Role Mapping* das *Refactoring* spezifizieren.

5.2 Beispielimplementierungen

Die Beispiele in diesem Kapitel zeigen, wie in *Refactory Quality Smells* implementiert werden können. Sie bestehen aus einem Muster zum Erkennen der Struktur des *Smells*, einem *Role Mapping*, dass dazu dient ein Strukturelement zu markieren und es an einen *Postprocessor*, zu übergeben, der das eigentliche *Refactoring* ausführt.

Um die *Refactoring*-Operationen und *Quality Smells* in *Eclipse* nutzen zu können, ist es notwendig ein *Eclipse Plugin* zu erstellen und die folgenden Erweiterungspunkte zu definieren:

- `org.emftext.refactoring.smell.registry.ui.calculation`
- `org.emftext.refactoring.rolemapping`
- `org.emftext.refactoring.postprocessor`

Daraufhin kann es als exportiertes Plugin installiert und die *Quality Smells* unter den Einstellungen entsprechend konfiguriert werden. So lässt sich unter anderem einstellen,

[3] https://www.eclipse.org/incquery/

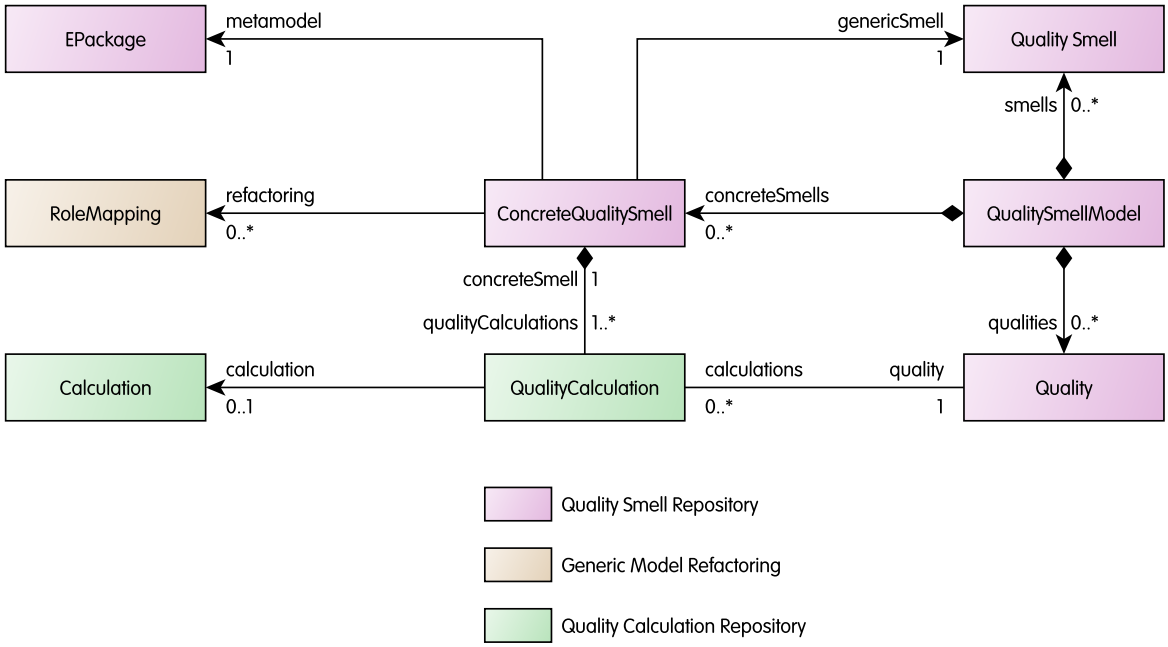


Abbildung 5.2: *Metamodell des Quality Smell Repository nach [RA13]*

welche Qualitäten wie stark beeinflusst werden. Tauchen während der Prüfung *Quality Smells* auf, werden diese im Editor angezeigt und können über einen *Quick Fix* refaktoriert werden (siehe Abbildung 5.3). Dieser *Quick Fix* wird automatischt angeboten, wenn ein *Refactoring* für einen *Quality Smell* registriert wurde.

```
public class TrackingHardwareIdTest extends Activity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);

        TelephonyManager tm = (TelephonyManager)
                getSystemService(Context.TELEPHONY_SERVICE);
        String id = tm.getDeviceId();
    }

}
```

The IMEI should not be used for security reasons.

1 quick fix available:

Uses a unique generated id for tracking the hardware id

Press 'F2' for focus

Abbildung 5.3: *Editor mit Markierung für einen* Quality Smell

5.2.1 Durable WakeLock

Das folgende *IncQuery Pattern* findet alle Aufrufe von `WakeLock.aquire()` innerhalb einer Klasse die von `Activity` erbt.

Code 5.2.1: IncQuery Pattern

```
pattern durableWakeLock(acquireExpression:MethodCall) = {
  // Die aktuelle Klasse soll von Activity erben.
  Class.^extends(actualClass, superClassRef);
  NamespaceClassifierReference.classifierReferences(superClassRef,
   classifierReference);
  ClassifierReference.target(classifierReference, superClass);
  Class.name(superClass, "Activity");

  // Die aktuelle Klasse besitzt eine Methode,
  Class.members(actualClass, anyMethod);
  // die direkt oder durch Untermethodenaufrufe eine Anweisung
   enthaelt,
  find parentContainsBindingExpression+(anyMethod, wakeLockStatement);

  // mit dem Namen aquire.
  ExpressionStatement.expression(wakeLockStatement, wakeLockIdentifier
   );
  IdentifierReference.next(wakeLockIdentifier, acquireExpression);
  MethodCall.target.name(acquireExpression, "acquire");

  // Diese Methode wird ohne Parameter aufgerufen
  neg find hasArguments(acquireExpression);

  // Und hat eine Referenz auf eine Variable,
  IdentifierReference.target(wakeLockIdentifier,
   wakeLockIdentifierReference);

  // die vom Typ WakeLock ist.
  LocalVariable.typeReference(wakeLockIdentifierReference,
   wakeLockIdentifierReferenceClassfier);
  NamespaceClassifierReference.classifierReferences.target.name(
   wakeLockIdentifierReferenceClassfier, "WakeLock");
}

private pattern parentContainsBindingExpression(parent, child) {
  StatementListContainer.statements(parent, child);
} or {
  ExpressionStatement.expression(parent, child);
} or {
  StatementContainer.statement(parent, child);
}

private pattern hasArguments(args) {
  MethodCall.arguments(args, _);
}
```

Das *Role Mapping* ist spezifisch für ein Metamodell. In unserem Beispiel nutzt es JaMoPP. Die Anweisung `maps <SelectX>` dient dazu ein definiertes Modellelement einem *Postprocessor* zu übergeben. Das ist notwendig da *Quality Smells* sehr plattformspezifisch sind.

Code 5.2.2: Role Mapping

```
ROLEMODELMAPPING FOR <http://www.emftext.org/java>

"Transforms acquire statement to use timeout" maps <SelectX> {
   Selection := references.MethodCall;
}
```

Code 5.2.3: Refactoring Postprocessor

```
public class AcquireWakeLockWithTimeout extends
    AbstractRefactoringPostProcessor {

  private final static String ACQUIRE_WITH_TIMEOUT = "60*1000*10";

  @Override
  public IStatus process(Map<Role, List<EObject>>
   roleRuntimeInstanceMap,
      EObject refactoredModel, ResourceSet resourceSet,
      ChangeDescription change, RefactoringSpecification refSpec,
      List<IModelRefactoringWizardPage> customWizardPages,
      boolean isFakeRun, Map<EObject, EObject> copier,
      List<? extends EObject> initialSelection) {

    // Nutze die Hilfsmethode, die den MethodCall aus dem RoleMapping
    extrahiert.
    MethodCall acquireMethodCall = RoleUtil.getFirstObjectForRole(
        "Selection", MethodCall.class, roleRuntimeInstanceMap);

    // Aendere den Namen der Methode.
    acquireMethodCall.getTarget().setName("aquireWithTimeout");

    // Erzeuge ein Modellfragment aus einer Zeichenkette,
    MultiplicativeExpression me = parsePartialFragment(
        ACQUIRE_WITH_TIMEOUT,
        ExpressionsPackage.Literals.MULTIPLICATIVE_EXPRESSION,
        MultiplicativeExpression.class).get(0);

    // um es dann als Argument hinzuzufuegen
    acquireMethodCall.getArguments().add(me);

    return Status.OK_STATUS;
  }

}
```

Der *Postprocessor* ersetzt den Aufruf von `WakeLock.aquire()` durch einen Timeout-basierten WakeLock: `WakeLock.aquireWithTimeout(60*1000*10)`.

Die Hilfsfunktion `parsePartialFragment()` zum Erstellen eines Modells aus einer Zeichenkette, ist in Anhang A.2 (Code A.2.1) zu finden.

5.2.2 No Low Memory Resolver

Das folgende *IncQuery Pattern* sucht, ob innerhalb einer Klasse die von `Activity` erbt, die Methode `onLowMemory()` implementiert wird.

Code 5.2.4: IncQuery Pattern

```
pattern noMemoryResolver(actualClass:Class) {
  Class.^extends(actualClass, superClassRef);
  NamespaceClassifierReference.classifierReferences(superClassRef,
   classifierReference);
  ClassifierReference.target(classifierReference, superClass);
  Class.name(superClass, "Activity");

  neg find hasMethod_mom(actualClass, "onLowMemoryResolver");
}

private pattern hasMethod_mom(class, method) = {
  Class.members.name(class, method);
}
```

Code 5.2.5: Role Mapping

```
ROLEMODELMAPPING FOR <http://www.emftext.org/java>

"Lets the activity class override onLowMemory() method" maps <SelectX>
    {
   Selection := classifiers.Class;
}
```

Der *Postprocessor* lässt die Klasse die Methode `onLowMemory()` implementieren. Die genaue Ausgestaltung, welche Ressourcen freigegeben werden, muss noch vom Softwareentwickler festgelegt werden.

Code 5.2.6: Refactoring Postprocessor

```
public class OverrideOnLowMemoryResolver extends
    AbstractRefactoringPostProcessor {

  private final static String ONLOWMEMORY =
      "\n" +
      "public void onLowMemory() {\n" +
      "\t// TODO Auto-generated method stub\n" +
      "\tsuper.onLowMemory();\n" +
      "};";

  @Override
  public IStatus process(Map<Role, List<EObject>>
   roleRuntimeInstanceMap,
      EObject refactoredModel, ResourceSet resourceSet,
      ChangeDescription change, RefactoringSpecification refSpec,
      List<IModelRefactoringWizardPage> customWizardPages,
      boolean isFakeRun, Map<EObject, EObject> copier,
      List<? extends EObject> initialSelection) {

    org.emftext.language.java.classifiers.Class activityClass =
   RoleUtil.getFirstObjectForRole(
        "Selection", org.emftext.language.java.classifiers.Class.class
    , roleRuntimeInstanceMap);

    ClassMethod  me = parsePartialFragment(
        ONLOWMEMORY,
        MembersPackage.Literals.CLASS_METHOD,
        ClassMethod.class).get(0);

    activityClass.getMembers().add(me);

    return Status.OK_STATUS;
  }

}
```

5.2.3 Interrupting From Background

Das folgende *IncQuery Pattern* sucht ob innerhalb einer Klasse die von `Service` oder `BroadcastReciever` erbt, eine *Activity* oder ein *Toast* gestartet wird.

Code 5.2.7: IncQuery Pattern

```
pattern interruptingFromBackground(expression:ExpressionStatement) {
    Class.^extends(actualClass, superClassRef);
    NamespaceClassifierReference.classifierReferences(superClassRef,
   classifierReference);
    ClassifierReference.target(classifierReference, superClass);
    find isServiceOrBroadcastReciever(superClass);

    find startsActivityOrToast(expression);
    Class.members(actualClass, method);
    find parentContainsSomething+(method, expression);
}

private pattern startsActivityOrToast(expression) {
  ExpressionStatement.expression(expression, startToastMethod);
  IdentifierReference.target.name(startToastMethod, "Toast");
  IdentifierReference.next(startToastMethod, toastExpression);
  MethodCall.target.name(toastExpression, "makeText");
  MethodCall.next(toastExpression, showToastExpression);
  MethodCall.target.name(showToastExpression, "show");
} or {
  ExpressionStatement.expression(expression, startActivitiyMethod);
  MethodCall.target.name(startActivitiyMethod, "startActivity");
}

private pattern isServiceOrBroadcastReciever(class) {
  find isClassOf(class, "Service");
} or {
  find isClassOf(class, "BroadcastReciever");
}

private pattern isClassOf(class, className) {
  Class.name(class, className);
}
```

Code 5.2.8: Role Mapping

```
ROLEMODELMAPPING FOR <http://www.emftext.org/java>

"Remove interrupt from background task" maps <SelectX> {
   Selection := statements.ExpressionStatement;
}
```

Das *Refactoring* entfernt die Methoden.

Code 5.2.9: Refactoring Postprocessor

```
public class RemoveInterrupFromBackground extends
    AbstractRefactoringPostProcessor {

  @Override
  public IStatus process(Map<Role, List<EObject>>
   roleRuntimeInstanceMap,
      EObject refactoredModel, ResourceSet resourceSet,
      ChangeDescription change, RefactoringSpecification refSpec,
      List<IModelRefactoringWizardPage> customWizardPages,
      boolean isFakeRun, Map<EObject, EObject> copier,
      List<? extends EObject> initialSelection) {


    ExpressionStatement expression = RoleUtil.getFirstObjectForRole("
   Selection",
        ExpressionStatement.class, roleRuntimeInstanceMap);

    ClassMethod parent = expression.getParentByType(ClassMethod.class)
   ;
    if(parent != null) {
      parent.getStatements().remove(expression);
    }

    return Status.OK_STATUS;
  }

}
```

5.2.4 Rigid AlarmManager

Das folgende *IncQuery Pattern* sucht ob innerhalb einer Klasse die von `Activity` erbt, ein exakter *AlarmManager* definiert wird.

Code 5.2.10: IncQuery Pattern

```
pattern rigidAlarmManager(acquireExpression:MethodCall) {
  Class.^extends(actualClass, superClassRef);
  NamespaceClassifierReference.classifierReferences(superClassRef,
   classifierReference);
  ClassifierReference.target(classifierReference, superClass);
  Class.name(superClass, "Activity");
  Class.members(actualClass, anyMethod);

  ExpressionStatement.expression(bindingStatement, wakeLockIdentifier)
   ;
  IdentifierReference.target(wakeLockIdentifier,
   wakeLockIdentifierReference);
  IdentifierReference.next(wakeLockIdentifier, acquireExpression);
  MethodCall.target.name(acquireExpression, "setRepeating");

  LocalVariable.typeReference(wakeLockIdentifierReference,
   wakeLockIdentifierReferenceClassfier);
  NamespaceClassifierReference.classifierReferences.target.name(
   wakeLockIdentifierReferenceClassfier, "AlarmManager");

  find parentContainsBindingExpression+(anyMethod, bindingStatement);
}
```

Code 5.2.11: Role Mapping

```
ROLEMODELMAPPING FOR <http://www.emftext.org/java>

"Replace exact AlarmManager with inexact" maps <SelectX> {
   Selection := references.MethodCall;
}
```

Der exakte *AlarmManager* wird durch einen inexakten ersetzt, womit Alarmierungen gebündelt werden können.

Code 5.2.12: Refactoring Postprocessor

```
public class SetInexactAlarmManager extends
                AbstractRefactoringPostProcessor {

  @Override
  public IStatus process(Map<Role, List<EObject>>
   roleRuntimeInstanceMap,
                EObject refactoredModel, ResourceSet resourceSet,
                ChangeDescription change, RefactoringSpecification
   refSpec,
                List<IModelRefactoringWizardPage> customWizardPages,
                boolean isFakeRun, Map<EObject, EObject> copier,
                List<? extends EObject> initialSelection) {


        MethodCall mc = RoleUtil.getFirstObjectForRole("Selection",
                        MethodCall.class, roleRuntimeInstanceMap);

        mc.getTarget().setName("setInexactRepeating");

        return Status.OK_STATUS;
  }
}
```

5.2.5 Tracking Hardware Id

Das folgende *IncQuery Pattern* sucht ob innerhalb einer Klasse die von `Activity` erbt, eine problematische Telefon-Id abgerufen wird.

Code 5.2.13: IncQuery Pattern

```
pattern trackingHardwareId(deviceIdIdentifier:IdentifierReference) {
  Class.^extends(actualClass, superClassRef);
  NamespaceClassifierReference.classifierReferences(superClassRef,
   classifierReference);
  ClassifierReference.target(classifierReference, superClass);
  Class.name(superClass, "Activity");
  Class.members(actualClass, anyMethod);

  MethodCall.target.name(deviceIdMethodExpression, "getDeviceId");

  IdentifierReference.target(deviceIdIdentifier,
   deviceIdIdentifierReference);
  IdentifierReference.next(deviceIdIdentifier,
   deviceIdMethodExpression);

  Variable.typeReference(deviceIdIdentifierReference,
   deviceIdIdentifierReferenceClassfier);
  NamespaceClassifierReference.classifierReferences.target.name(
   deviceIdIdentifierReferenceClassfier, "TelephonyManager");

  find parentContainsBindingExpression+(anyMethod, bindingStatement);
}
```

Code 5.2.14: Role Mapping

```
ROLEMODELMAPPING FOR <http://www.emftext.org/java>

"Use a unique generated id for tracking the hardware id" maps <SelectX
    > {
   Selection := references.IdentifierReference;
}
```

Der problematische Aufruf wird durch einen empfehlenswerteren ersetzt.

Code 5.2.15: Refactoring Postprocessor

```
public class UseUUID extends AbstractRefactoringPostProcessor {

  private static final String UUID = " Settings.Secure.ANDROID_ID;";

  @Override
  public IStatus process(Map<Role, List<EObject>>
   roleRuntimeInstanceMap,
      EObject refactoredModel, ResourceSet resourceSet,
      ChangeDescription change, RefactoringSpecification refSpec,
      List<IModelRefactoringWizardPage> customWizardPages,
      boolean isFakeRun, Map<EObject, EObject> copier,
      List<? extends EObject> initialSelection) {

    IdentifierReference ir = RoleUtil.getFirstObjectForRole(
      "Selection", IdentifierReference.class, roleRuntimeInstanceMap);

    CompilationUnit cunit = ir.getParentByType(CompilationUnit.class);
    if (cunit != null) {
      addImport("android.provider.Settings", cunit);
    }

    IdentifierReference fragment = parsePartialFragment(UUID,
        ReferencesPackage.Literals.IDENTIFIER_REFERENCE,
        IdentifierReference.class).get(0);
    LocalVariable st = (LocalVariable) ir.eContainer();
    st.setInitialValue(fragment);

    return Status.OK_STATUS;
  }

  private void addImport(String clazzName, CompilationUnit cunit) {
    cunit.getImports().add(
        parsePartialFragment("import " + clazzName,
            ImportsPackage.Literals.CLASSIFIER_IMPORT,
            ClassifierImport.class).get(0));
  }

}
```

6 Zusammenfassung & Ausblick

6.1 Zusammenfassung

In dieser Arbeit wurde geschildert, dass Smartphones omnipräsent sind und Anwender durch unterschiedliche Hardwarespezifikationen und -restriktionen hohe Anforderungen an die Qualität der Software stellen. Deswegen wird während des Softwareentwicklungszyklus die Anwendung dahingehend optimiert, dass sie Qualitätsanforderungen in höherem Maße erfüllen. Dafür wird das Softwaresystem nach *Bad Smells* durchsucht und betreffende Stellen refaktoriert. Weil den Entwicklern aber der direkte Bezug zwischen *Bad Smells* und nichtfunktionalen Anforderungen fehlt, was ein qualitätsorientiertes Arbeiten erschwert, wurde das Konzept der *Quality Smells* von Reimann und Aßmann [RA13] genutzt, welches die Konzepte der *Bad Smells*, des *Refactorings* und der Softwarequalitäten vereint. Damit ist es möglich, eine – bisher fehlende – Werkzeugunterstützung, zu erstellen. Als Grundlage dafür, wurde eine Entwicklerstudie durchgeführt, welche aus verschiedenen Internetquellen *Quality Smells* der Android Plattform identifiziert. Die gefundenen *Bad Smells* wurden dabei in ein eigens entwickeltes Schema konvertiert und in den Katalog eingefügt. Darauf aufbauend wurde gezeigt, wie aus diesem Katalog und dem Programm *Refactory* eine Werkzeugunterstützung für die Erkennung und Behebung von *Quality Smells* konstruiert werden kann.

6.2 Ausblick

Um den Katalog zu erweitern und so den Nutzen zu erhöhen, ist es notwendig, weitere *Quality Smells* zu identifizieren, zum Beispiel durch weitere Entwicklerstudien oder durch das Portieren bekannter *Bad Smells* von anderen Plattformen. Auf diese Weise ist es auch möglich einen Katalog von generischen *Quality Smells* zu erzeugen. *Quality Smells* sind von Natur aus plattformspezifisch in Bezug auf Hardware, Programmiersprache oder Framework, jedoch haben einige *Smells* eine allgemeinere Wahrheit. Beispielsweise zeigt der Smell Data Transmission Without Compression, dass es generell energieeffizienter ist, Daten komprimiert über das Netzwerk zu übertragen. Solche generischen *Quality Smells* erlauben die Portierung und Anwendung auf verschiedenen Plattformen. Das Hinzufügen neuer *Bad Smells* kann es erforderlich machen, das Schema zu erweitern, zum Beispiel um die Eigenschaften „Bekannte Ausnahmen“, „Bekannt als“, „Bekannte Verwendung“, „Refaktorierungsumfang“ (Idiom, Mikroarchitektur) oder „Metrik“.

Ein weiterer Aspekt der untersucht werden sollte, gerade hinsichtlich unterschiedlicher Dimensionen des *Refactorings* [ASH13], ist der Kosten-Nutzen Faktor. Zum Beispiel ist die Forderung nach guter Wartbarkeit eigentlich ein Wunsch nach geringen Wartungskosten. Auch muss es nicht notwendig sein, die Effizienz eines Algorithmus zu erhöhen, wenn ausreichende Effektivität, zum Beispiel durch schnellere Hardware, zu realisieren ist.

In dieser Arbeit haben wir aus den Erfahrungen der Entwickler *Quality Smells* abgeleitet. Will man jedoch von einer transzendenten (heuristischen), hin zu einer produktorien-

tierten Sicht auf Qualitäten, ist es wichtig diese Verbindung genauer zu untersuchen. Das ist mit dem in Kapitel 2.2 vorgestellten Qualitätsmodell jedoch nicht möglich. Es muss geprüft werden, ob der Ansatz von [Dei09a] auch auf andere SQ erweitert werden kann.

In Tabelle 4.3, über betroffene Qualitäten von *Quality Smells*, zeigen sich verschiedene Abhängigkeiten zwischen Qualitäten. Beispielsweise kommt Sicherheit und Benutzerkonformität in drei von vier Fällen zusammen vor. Solche Abhängigkeiten lassen sich durch die Untersuchung weiterer *Quality Smells* identifizieren. Ob sich diese Abhängigkeiten jedoch lediglich auf Plattform- beziehungsweise Implementierungsebene definieren lassen, oder auch allgemeiner, muss untersucht werden. Dass die Startzeit einer Anwendung aber auch die Benutzbarkeit erhöht, zeigt, dass es zumindest auf einige Softwarequalitäten zutrifft.

Ein mehrdimensionales Qualitätsmodell [Dei09a; Gar84], das verschiedene Aspekte wie Rolle, Phase, Kosten/Aufwand, Metrik und Artefakt berücksichtigt, wäre zum einen für die Spezifikation von Anforderungsdokumenten nützlich, aber auch für die umfassende Definition einer Ontologie für *Quality Smells*. Die formale Definition von Beziehungen und Attributen von *Quality Smells* in einer Ontologie, erlaubt eine einheitliche Kommunikation und intuitive Suche durch logisches Schließen. Bisherige Ontologien, wie die der verwandten *Design Patterns* [Kam07], lassen den Aspekt Qualität jedoch unbeachtet oder verweisen nur implizit darauf, zum Beispiel durch die Einhaltung objektorientierter Prinzipien. Das zeigt, dass eine vollständige Definition von Softwarequalitäten in einem Qualitätsmodell eine notwendige Voraussetzung ist, um *Quality Smells* exakt abzubilden. Dabei sind die in unserem Schema vorgestellten Attribute (Kapitel 3.2), gegebenenfalls um weitere zu ergänzen.

A Anhang

A.1 Filterwörter

energy, power, drain, efficient, inefficient, inefficiency, efficiency, energy consumption, power consumption, memory consumption, energy bug, power bug, energy drain, performance, io access, speed up, code quality, cleen code, response time, faster, memory, overhead, code review, wake lock, refactoring, refactor, blocking operation, best practice, code smell, bad smell, access time, traffic, reduce, power-saving, power saving, energy-saving, energy saving, time-saving, time saving, low battery, economic, battery, periodic, minimize, consumption, high usage, slow, leak, power managment, power save, discharge, uncharge

A.2 Hilfsfunktionen für IncQuery Pattern

Code A.2.1: Methode zum Erzeugen eines Modellfragments aus einer Zeichenkette.

```
/**
 * Parses the given <code>fragment</code> and returns the according
 * model  elements. The parameter <code>startRule</code> specifies
 * the {@link EClass} which should be the starting rule.
 */
public static <RuleType extends EObject> List<RuleType>
    parsePartialFragment(
       String fragment, EClass startRule, Class<RuleType> type) {
  URI uri = URI.createURI("temp.java");
  ResourceSet resourceSet = new ResourceSetImpl();
  Resource resource = resourceSet.createResource(uri);
  ByteArrayInputStream inputStream = new ByteArrayInputStream(
      fragment.getBytes());
  Map<?, ?> typeOption = Collections.singletonMap(
      IJavaOptions.RESOURCE_CONTENT_TYPE, startRule);
  try {
    resource.load(inputStream, typeOption);
  } catch (IOException e) {
    e.printStackTrace();
  }
  List<RuleType> result = new ArrayList<RuleType>();
  for (EObject parseResult : resource.getContents()) {
    if (type.isInstance(parseResult)) {
      result.add(type.cast(parseResult));
    }
  }
  return result;
}
```

Literatur

[ASH13] Paris Avgeriou, Michael Stal und Rich Hilliard. "Architecture Sustainability". en. In: *IEEE Software* 30.6 (Nov. 2013), S. 40–44. ISSN: 0740-7459. DOI: 10.1109/MS.2013.120. URL: www.computer.org/csdl/mags/so/2013/06/mso2013060040.html (siehe S. 13, 109).

[Bac96] James Bach. "The challenge of "good enoughßoftware". In: (1996). URL: http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.10.2499 (siehe S. 8).

[Bal09] Helmut Balzert. *Lehrbuch Der Softwaretechnik: Basiskonzepte Und Requirements Engineering*. Springer DE, 2009, S. 624. ISBN: 3827422477. URL: http://books.google.com/books?id=vmfIb9R2QikC&pgis=1 (siehe S. 2).

[Bec07] Kent Beck. *Implementation Pattern*. Pearson Education, 2007, S. 176. ISBN: 013270255X. URL: http://books.google.com/books?id=xLyXPCxBhQUC&pgis=1 (siehe S. 12).

[Bro98] William J. Brown. *AntiPatterns: Refactoring Software, Architectures, and Projects in Crisis*. Wiley, 1998, S. 309. ISBN: 0471197130. URL: http://books.google.com/books?id=Dp9QAAAAMAAJ&pgis=1 (siehe S. 11, 19).

[Com13] Kantar Worldpanel ComTech. *Smartphone-Betriebssysteme - Marktanteile am Absatz in Deutschland bis 2013 — Statistik*. 2013. URL: http://de.statista.com/statistik/daten/studie/225381/ (siehe S. 2).

[Dei09a] Florian Deißenböck. "Continuous quality control of long lived software systems". In: (2009). URL: http://d-nb.info/998600407/ (siehe S. 11, 13, 110).

[Dei09b] Florian Deißenböck. "Kontinuierliches Qualitäts-Controlling langlebiger Softwaresysteme." In: *Ausgezeichnete Informatikdissertationen* (2009). URL: https://wwwbroy.in.tum.de/~deissenb/publications/2010_deissenboeckf_diss_summary.pdf (siehe S. 10).

[Dra12] Dratio Internet Data Service. *Gründe für die Deinstallierung von Apps in China im Jahr 2012 — Statistik*. 2012. URL: http://de.statista.com/statistik/daten/studie/240442 (siehe S. 1).

[Fow+12] Martin Fowler u. a. *Refactoring: Improving the Design of Existing Code*. Addison-Wesley, 2012, S. 455. ISBN: 013306526X. URL: http://books.google.com/books?id=HmrDHwgkbPsC&pgis=1 (siehe S. 2, 12, 19).

[Fow99] Martin Fowler. *Refactoring: Improving the Design of Existing Code*. Addison-Wesley, 1999 (siehe S. 11, 15, 23).

[Gam+94] Erich Gamma u. a. *Design Patterns: Elements of Reusable Object-Oriented Software*. Pearson Education, 1994. ISBN: 0321700694. URL: http://books.google.com/books?id=6oHuKQe3TjQC&pgis=1 (siehe S. 12, 19).

[Gar+09] J. Garcia u. a. “Identifying Architectural Bad Smells”. In: *13th European Conference on Software Maintenance and Reengineering*. 2009. DOI: 10.1109/CSMR.2009.59 (siehe S. 12).

[Gar84] David A. Garvin. *What Does “Product Quality” Really Mean?* 1984. URL: http://sloanreview.mit.edu/article/what-does-product-quality-really-mean/ (siehe S. 8, 9, 110).

[Geb11] Michael Gebhart. *Qualitätsorientierter Entwurf von Anwendungsdiensten*. KIT Scientific Publishing, 2011, S. 319. ISBN: 3866447043. URL: http://books.google.com/books?id=et6xIjiRQygC&pgis=1 (siehe S. 9).

[Got+12] Marion Gottschalk u. a. “Removing Energy Code Smells with Reengineering Services”. In: *Beitragsband der 42. Jahrestagung der Gesellschaft für Informatik e.V.* 2012 (siehe S. 12, 36).

[Gui12] Hervé Guihot. *Pro Android Apps Performance Optimization*. Apress, 2012. ISBN: 1430239999. DOI: 10.1007/978-1-4302-4000-6. URL: http://books.google.com/books?hl=en&lr=&id=7k_beXmqgVoC&oi=fnd&pg=PA1&dq=Pro+Android+Apps+Performance+Optimization&ots=Pci4Gns274&sig=bs7mq1-654aicEaaenDx0mMuExo (siehe S. 16, 35, 41, 61, 65, 86).

[HB10] Hagen Höpfner und Christian Bunse. “Towards an Energy-Consumption Based Complexity Classification for Resource Substitution Strategies”. In: *Proceedings of the 22nd Workshop "Grundlagen von Datenbanken 2010"*. 2010 (siehe S. 2, 12).

[Kam07] Holger Kampffmeyer. *Formalization of Design Patterns by Means of Ontologies*. 2007. URL: http://books.google.com/books?hl=en&lr=&id=d3fRh__yABEC&oi=fnd&pg=PA1&dq=Formalization+of+Design+Patterns+by+Means+of+Ontologies&ots=mX5QuEn36B&sig=YAJO_Mamo_UYIjScbmem1jPbBDw (siehe S. 110).

[Leh80] Meir M. Lehman. “On understanding laws, evolution, and conservation in the large-program life cycle”. In: *Journal of Systems and Software* 1 (1980), S. 213–221 (siehe S. 13).

[Lig09] Peter Liggesmeyer. *Software-Qualität: Testen, Analysieren und Verifizieren von Software*. Springer, 2009, S. 526. ISBN: 3827420563. URL: http://books.google.com/books?id=-eoerZI2fYIC&pgis=1 (siehe S. 8).

[McC76] TJ McCabe. “A complexity measure”. In: *Software Engineering, IEEE Transactions on* 4 (1976), S. 308–320. URL: http://scholar.google.com/scholar?hl=en&btnG=Search&q=intitle:THOMAS+J.+McCABE#5 (siehe S. 9).

[Mob13] ComScore MobiLens. *Anzahl der Smartphone-Nutzer in Deutschland bis 2013 — Statistik*. 2013. URL: http://de.statista.com/statistik/daten/studie/198959/ (siehe S. 1).

[MTM07] Tom Mens, Gabriele Taentzer und D Müller. “Challenges in model refactoring”. In: *Proc. 1st Workshop on Refactoring ...* (2007), S. 1–5. URL: http://www.researchgate.net/publication/228654952_Challenges_in_model_refactoring/file/79e415093d7c73153a.pdf (siehe S. 11).

[ÖP05] Gunnar Övergaard und Karin Palmkvist. *Use cases: patterns and blueprints*. Addison-Wesley, 2005, S. 434. ISBN: 0131451340. URL: http://books.google.com/books?id=-cxQAAAAMAAJ&pgis=1 (siehe S. 11).

[Par94] David Lorge Parnas. "Software aging". In: (Mai 1994), S. 279–287. URL: http://dl.acm.org/citation.cfm?id=257734.257788 (siehe S. 13).

[Pat+12] Abhinav Pathak u. a. "What is keeping my phone awake?: characterizing and detecting no-sleep energy bugs in smartphone apps". In: *Proceedings of the 10th ...* MobiSys '12 (2012), S. 267–280. DOI: 10.1145/2307636.2307661. URL: http://doi.acm.org/10.1145/2307636.2307661 (siehe S. 12).

[PK13] Gustavo H. Pinto und Fernando Kamei. "What programmers say about refactoring tools?" In: *Proceedings of the 2013 ACM workshop on Workshop on refactoring tools - WRT '13*. New York, New York, USA: ACM Press, Okt. 2013, S. 33–36. ISBN: 9781450326049. DOI: 10.1145/2541348.2541357. URL: http://dl.acm.org/citation.cfm?id=2541348.2541357 (siehe S. 12).

[RA13] Jan Reimann und Uwe Aßmann. "Quality-Aware Refactoring For Early Detection And Resolution Of Energy Deficiencies". In: *Proceedings of 4th International Workshop on Green and Cloud Computing Management*. 2013 (siehe S. 2, 12, 95, 97, 109).

[Rei10] Jan Reimann. "Generisches Modellrefactoring für EMFText". Magisterarb. Technische Universität Dresden, 2010 (siehe S. 95).

[RG98] Dirk Riehle und Thomas Gross. "Role Model Based Framework Design and Integration". In: *Proceedings of OOPSLA '98*. Vancouver, British Columbia, Canada: ACM, 1998. ISBN: 1-58113-005-8. DOI: http://doi.acm.org/10.1145/286936.286951 (siehe S. 95).

[RSA12] Jan Reimann, Mirko Seifert und Uwe Aßmann. "On the reuse and recommendation of model refactoring specifications". In: *Software & Systems Modeling* 12.3 (Apr. 2012), S. 579–596. ISSN: 1619-1366. DOI: 10.1007/s10270-012-0243-2. URL: http://link.springer.com/10.1007/s10270-012-0243-2 (siehe S. 96).

[SW03] Connie U. Smith und Lloyd G. Williams. "More New Software Performance Antipatterns: Even More Ways to Shoot Yourself in the Foot". In: *Computer Measurement Group Conference 2003*. Computer Measurement Group. 2003, S. 717–725 (siehe S. 12).

[Vet+13] Antonio Vetro' u. a. "Definition, Implementation and Validation of Energy Code Smells: an Exploratory Study on an Embedded System". In: *ENERGY 2013, The Third International Conference on Smart Grids, Green Communications and IT Energy-aware Technologies*. Hrsg. von Steffen Fries und Petre Dini. Lisbon, Portugal, März 2013 (siehe S. 12).

[WGR13] Claas Wilke, Sebastian Götz und Sebastian Richly. "JouleUnit: a generic framework for software energy profiling and testing". In: *Proceedings of the 2013 workshop on Green in/by software engineering*. Fukuoka, Japan, 2013. ISBN: 978-1-4503-1866-2. DOI: 10.1145/2451605.2451610 (siehe S. 9).

[Wik13a] Wikipedia. *Android (Betriebssystem)*. de. Dez. 2013. URL: http://de.wikipedia.org/w/index.php?title=Android_(Betriebssystem)&oldid=125268972 (siehe S. 5).

[Wik13b] Wikipedia. *Best practice – Wikipedia*. 2013. URL: http://de.wikipedia.org/wiki/Best_practice (siehe S. ix).

[Wil+13] Claas Wilke u. a. "Comparing mobile applications' energy consumption". In: SAC '13. New York, NY, USA: ACM, 2013, S. 1177–1179. ISBN: 978-1-4503-1656-9. DOI: 10.1145/2480362.2480583. URL: http://doi.acm.org/10.1145/2480362.2480583 (siehe S. 1).

Akronyme

E | **M** | **S** | **U** | **X**

E

EMF

Ein Open-Source Framework zum automatisierten Erzeugen, Manipulieren und Abfragen von strukturierten Modellen. Siehe `http://www.eclipse.org/modeling/emf/`. 95

M

MA

mobile Anwendung. 1, 2, 5, 6, 15, 20, *siehe Glossar:* MOBILE ANWENDUNG

MDSD

Model Driven Software Development – Modellgetriebene Softwareentwicklung. 95

S

SQ

Softwarequalität. 2, 6, 9, 11, 12, 15, 19, 21, 97, *siehe Glossar:* SOFTWAREQUALITÄT

SQuaRE

Systems and Software Quality Requirements and Evaluation. 8

U

UI

User Interface. 6, 19, *siehe Glossar:* USER INTERFACE

UML

Unified Modelling Language. *siehe Glossar:* UML

X

XML

Extensible Markup Language. vi, x, *siehe Glossar:* XML

Glossar

A | B | M | Q | R | S | U | W | X

A

API

Das *Application Programming Interface* beschreibt eine Schnittstelle eines Softwaresystems, das anderen System zur Verfügung gestellt wird.. 16, 17

B

Bad Smell

Auch *Code Smell*. So wird ein Codeartefakt genannt, dass durch *Refactorings* verbessert werden kann, um so eine Anforderung besser zu erfüllen.. ix, 2, 3, 6, 7, 11, 12, 16, 21, 97

Best Practices

"bezeichnet bewährte, optimale bzw. vorbildliche Methoden, Praktiken oder Vorgehensweisen"([Wik13b]). 2, 16

Bug-Tracker

Eine Software, die für die Erfassung und Dokumentation von Programmfehlern genutzt wird. 2, 15, 16

M

Mobile Anwendung

Eine Anwendung die auf einem mobilen Gerät läuft. vi

Q

Quality Smell

Ein Konzept zur Beschreibung von *Bad Smells*, in dem zudem der Einfluss auf Qualitäten betrachten wird und *Refactorings* angeboten werden. 2, 3, 12, 13, 15, 19–21, 24, 25, 97, 98

R

Refactoring

Bezeichnet die Umstrukturierung von Code.. ix, 2, 12, 13, 19, 21, 95–97

S

Softwarequalität

Eine nicht-funktionale Anforderung in der Softwareentwicklung. vi, 2, 9, 11, 12, 97

U

UML

Unified Modelling Language, grafische Modellierungssprache für Softwaresysteme. 12, 95

User Interface

Benutzeroberfläche. vi

User-Agent

Ein Kürzel zur Identifizierung des Internetbrowsers. 18

W

Webcrawler

Ein Programm, dass Webseiten automatisch durchsucht und analysiert. Auch *Webspider* oder *Webscraper* gennant. 17

X

XML

Ist eine Auszeichnungssprache für hierarchisch srukturierte Daten in Textdokumenten. vi

XPath

Ist eine Abfragesprache für XML-Dokumente. 17

Erklärung

Ich erkläre, dass ich die vorliegende Arbeit selbstständig, unter Angabe aller Zitate und nur unter Verwendung der angegebenen Literatur und Hilfsmittel angefertigt habe.

Dresden, den 27. März 2014