

CCD ein Praxisbericht

Thema:

Praktische Anwendung von CCD mit einer integrierten Workbench

Jörg Rückert



clean-code-developer.de

Prinzipien, Regeln und Praktiken für bessere Software

Hintergrund und Bekenntnis

Das Thema CCD wird von Softwareentwicklern angenommen und diskutiert. Dies kann man in Foren aktiv verfolgen. Interessant dabei ist, welche unterschiedlichen Erfahrungen einzelne Entwickler gemacht haben und wie schrittweise an das Thema herangegangen wird.

Ich persönlich habe in einem Projekt zu dem Thema gefunden, bei dem CCD aktiv gelebt wurde.

Mir war allerdings während des Projektes gar nicht bewusst gewesen, dass ich mich bereits im CCD-Umfeld bewegte. Gewundert hatte mich nur immer wieder, wie professionell das Team mit Softwareänderungen umgegangen ist.

Grundlegende Prinzipien hörte ich damals zum ersten Mal, obwohl ich diese teilweise schon

angewendet hatte, aber nicht zuordnen konnte. Die Flashbacks setzten später ein: „Transitive Navigation ist verpönt und hat für mich nun mit dem Law of Demeter (LoD) auch einen Namen“. Die Faszination für CCD hat mich seither nicht mehr losgelassen, ich studierte „Clean Code“ von Robert C. Martin und weitere Literatur, die sich mit der Softwarewartung beschäftigt.

Die CCD-Initiative von Ralf Westphal und Stefan Lieser hatte mich, nachdem ich über deren Webseite gestolpert war, sofort positiv angesprochen und gab mir den Anreiz, mich noch intensiver mit dem Thema auseinanderzusetzen. Ich wollte CCD nicht nur theoretisch kennenlernen und verfolgen, sondern gezielt nutzen.

Nach dem Bekenntnis zu CCD und den Aha-Erlebnissen liefen

die ersten Programmiersessions teilweise noch unstrukturiert ab. Ich habe es von Anfang an gemocht die Pfadfinderregel anzuwenden, hatte aber mit anderen Prinzipien immer wieder das Problem die Theorie mit der Praxis zu verbinden. Ähnlich ging es mir am Anfang bei der Anwendung von Patterns. Patterns theoretisch zu kennen ist eine Sache, deren sinnvollen Einsatz in einer Quellcodebasis zu erkennen, eine ganz andere. Man benötigt Pattern-Erfahrung, die Zeit für den Lernprozess und Training.

Ich wendete bereits Werkzeuge, wie PMD und Findbugs an und bin mit dem JUnit-Plugin gut vertraut. Mir fehlte aber eine integrierte Workbench, in der ich CCD-Einheiten strukturiert planen und mit voller Aufmerksamkeit anwenden konnte.

Idee und Aufwärmphase

Ich recherchierte zunächst und habe einzelne für CCD brauchbare Werkzeuge gefunden, die ich teilweise schon kannte. Ich wollte allerdings keine Werkzeugsammlung, sondern eine integrierte Workbench, in der ich CCD-Einheiten planen und ausführen konnte. Die Entscheidung fiel deshalb nicht schwer, eine eigene Workbench zu programmieren.

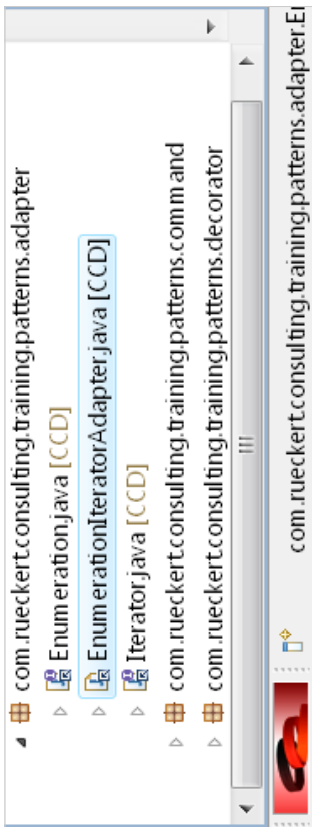
Nach der Anforderungsanalyse haben sich für die Workbench folgende Schlüsselpunkte ergeben: Eine Wissensbasis in Form eines Spickzettels, die Anzeige des Grades, einen Inspector und in der Konsequenz einen Analyzer.

Als Basis für die Entwicklung sollte die geschätzte Entwicklungsumgebung Eclipse dienen.

Das Ergebnis der Entwicklung sollte eine praxistaugliche Workbench für meine CCD Aktivitäten werden. Die Projekterfahrung mit SWT/JFace im Rahmen der Plugin-Programmierung hatte ich bereits, sodass der Spickzettel in zwei Sprints implementiert war. Bei der Verwaltung und Anzeige des Grades war mehr Denkarbeit nötig, das schadete aber nicht.

CCD ein Praxisbericht

Die Sprints zum CCD Inspector



„Sauberen Quellcode zu schreiben ist harte Arbeit. Es erfordert mehr als nur Prinzipien und Patterns zu kennen.“

Ein weiterer Sprint und die Grad-Anzeige war implementiert. Schnell mal in Eclipse als Feature rumgespielt, alles läuft wunderbar. Nun waren es schon zwei Plugins unter dem Dach des Features. Zweifel: Soll ich weiter laufen, lohnt sich der Aufwand einen Inspector zu programmieren? Wie soll der Inspector im Detail überhaupt aussehen?

Ich möchte zunächst nur mal eine Anzeige haben und meine Quellen, die unter CCD stehen in einer eigenen View verwalten.

Die Quelldateien, die ich auswähle, sollen dekoriert sein, so wie man das von CVS und SVN kennt. Ich möchte einen Status vergeben können und beim Review der Quellen gezielt Prinzipien und Praktiken über einen Dialog zuordnen. Die Plugins, die ich zusätzlich zu dem Inspector verwende, sollen in Kombination mit dem Inspector in einer CCD Perspektive untergebracht werden. Durch die Kombination der Plugins in einer gemeinsamen Perspektive, ist ein gezieltes CCD konformes Arbeiten möglich.

Die Details waren durchdacht, skizziert und wurden in mehreren Sprints programmiert. Mit dem Ergebnis, CCD nun endlich strukturiert in einer eigenen Perspektive in Kombination mit den mir bereits vertrauten Plugins anwenden zu können. Der erste Meilenstein war erreicht. Es war allerdings noch die Rede von einem Analyzer. Weiteres Abwägen, was bringt der Analyzer für Vorteile, lohnt sich der Aufwand, neue Anforderungen, weitere Sprints. Nochmal scharf nachdenken: Was soll der Analyzer im Detail leisten?

Der Weg zum CCD Analyzer

Semantik, die bei CCD häufig anzutreffen ist, kann sicherlich nicht in einfacher Form programmiert werden. Es bleiben also nur noch die syntaktische Prüfungen übrig, die der Java Compiler und mir vertraute Plugins von Hause aus nicht unterstützen.

Wieder nachdenken: Wie bilde ich die syntaktischen Regeln ab? Baue ich einen Parser oder gehe ich mit regulären Ausdrücken an die

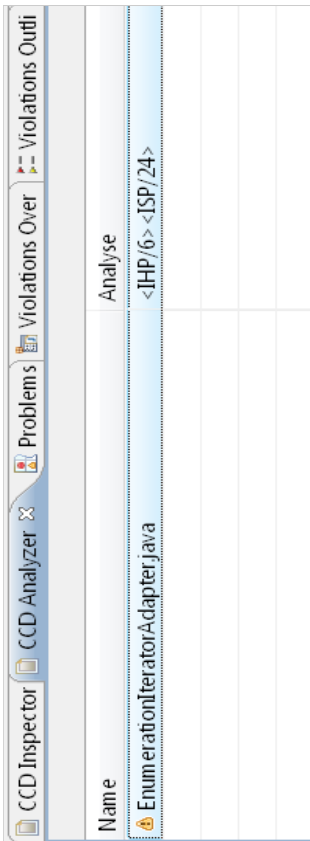
Aufgabenstellung heran? KISS! Reguläre Ausdrücke sind zur Lösung der Aufgabe gut genug. Zunächst einmal drei Regeln bauen, um die transitive Navigation (LoD), den Verstoß gegen IHP (öffentliche Felder) und Tell don't ask (keine Getter-Methoden) aufzustöbern. Die JUnit-Tests laufen schon und die Analyzer View ist im Anschluss zeitnah implementiert.

Die Analyzer View zeigt die Verletzung von CCD Prinzipien an. Beim Aufschalten des Quellcode-Editors, werden die Zeilen, die einen Verstoß beinhalten hervorgehoben und annotiert. Durch die Annotierung wird sofort klar, in welchem Quellcodeabschnitt das Problem liegt. Um den Überblick nicht zu verlieren ist die Quellcodedatei zusätzlich auch noch im „Package Explorer“ markiert.

Name	Status	Arbeitspaket
Enumeration.java	bearbeitet	
EnumerationIterator...	in Arbeit	Prinzipien: ISP,IHP
Iterator.java	bearbeitet	

CCD ein Praxisbericht

Das Zusammenspiel von CCD Inspector und Analyzer



Der Inspector verwaltet die Quellcodedateien die unter CCD stehen. Ein Bündel von Quellcodedateien wird dabei per Drag-and-Drop vom „Package Explorer“ in die Inspector View gezogen. Der nächste Schritt in der Verarbeitungskette ist das Umsetzen des Status und darauf folgend das Review der Quellcodedateien. Der Reviewer wählt dabei dialogorientiert offensichtliche Verstöße gegen CCD Prinzipien aus und notiert Handlungsanweisungen zur Beseitigung der erkannten Probleme. Der Analyzer unterstützt den Reviewer bei

seiner Arbeit indem er anhand hinterlegter Regeln automatisch die Verletzung von CCD Prinzipien identifiziert.

Der Reviewer selektiert ein Bündel von Quellcodedateien in der Inspector View und aktiviert per Kontextmenü die Analyse. Erkannte Probleme werden vom Analyzer klassifiziert und mit Zeilennummern versehen visualisiert. Per Doppelklick in der Analyzer View schaltet man die entsprechende Quellcodedatei auf, um die angezeigten Problemstellen zu inspizieren. Der Analy-

zer unterstützt in der Basisversion die Suche nach Prinzipverletzungen folgender Kategorien: IHP, ISP, LSP, LoD und Tell don't ask. Das Finden von Quellcode-duplikaten (DRY) wird bereits durch PMD abgedeckt und ist deshalb keine Funktionalität des Analyzers. Weitere Regeln sind integrierbar. Es ist deshalb vorgesehen auch allgemeine Refactoring Regeln zu hinterlegen, die auskommentierten Source Code, blinde Kommentare, Boolean Parameter in Methodensignaturen und überlange Methodensignaturen betreffen.

Ausblick und Fazit

Die CCD Workbench erleichtert das Erkennen von Verletzungen einzelner CCD Prinzipien und unterstützt gezielt den Reviewprozess. Es besteht sogar die Möglichkeit Regelverletzungen per Quick-Fix zu reparieren. Neben den Regeln zur Erkennung von Regelverletzungen werden dabei auch Regeln für die Reparatur integriert. Die Realisierung des Vier-Augen-Prinzips wird Bestandteil der Workbench wer-

den. Der zum inspizierten Quellcode erstellte Bericht ist dann dem Autor des Quellcodes für die nachfolgende Korrektur zeigbar. Man bedenke, dass semantisch bedingte Regelverletzungen nicht per Quick-Fix korrigierbar sind. Für die schnelle Navigation in der Quellcodedatei wird ein Instrument installiert, das die zeilenweise Navigation zu Regelverletzungen unterstützt.

Die Kombination des Inspectors mit dem Analyzer und den hinterlegten Basisregeln bewährt sich gut und ergänzt die Plugins der CCD Perspektive. Die Workbench ist in der Basisversion programmiert und entwickelt sich stetig weiter.

Fazit: Lohnt sich der Aufwand eine Workbench zu implementieren?! Ich meine ja, und sei es nur der Programmierfreude wegen.

„Das tägliche Reflektieren alleine genügt nicht. Erkannte Probleme sind konform der Pfadfinderregel möglichst zeitnah zu beheben.“

