

eclipse

MAGAZIN

www.eclipse-magazin.de

EXKLUSIV AUF CD:

Saros - Distributed Party Programming

Ein Plug-in für gemeinsames Editieren

AnyEdit

Das Schweizer Taschenmesser unter den Eclipse-Plug-ins

EditBox

Erweiterte Codeformatierung und Visualisierung in Eclipse

QuickRex

Reguläre Ausdrücke innerhalb des Eclipse SDK prüfen

Yari

Yet Another RCP Inspector

REVISITED

Plug-in-Parade

Top Ten der nützlichen Eclipse-Helferlein

► **Eclipse für iPhone** >> 68

iPhonical - DSL 4 Mobile

► **IDE 2.0?** >> 92

Kollaborativ zur IDE der Zukunft

► **Deklarative UIs in e4** >> 19

XWT versus Toolkit Model



JAHRES-ARCHIV 2009

Jetzt für Abonnenten kostenlos online >> S. 2

► **Visual Editor for XML**

XML-Dateien wie mit einer Textverarbeitung editieren >> 13

► **Eclipse Target Platform**

Neue Serie: Eclipse effizient als Zielplattform nutzen >> 8



Datenträger enthält Info- und Lehrprogramme gemäß §14 JuSchG



Unmittelbares Feedback mit den Plug-ins von projectusus.org

Das ist hier Usus...

» LEIF FRENZEL, NICOLE RAUCH UND STEFAN SCHÜRLE

Gut eingespielte Teams entwickeln oft einen Usus im Projekt - eine allseits geteilte Auffassung darüber, „wie hier bei uns programmiert wird“. Teil dieser Kultur ist üblicherweise neben einer Sammlung von Werkzeugen und ihrer Einstellungen auch eine anerkannte und gelebte Coding- und Refactoring-Praxis. Wir stellen Plug-ins zur Unterstützung einer solchen gemeinsamen Praxis vor, die wir am Usus unserer Projekterfahrungen modelliert und in ein Open-Source-Projekt überführt haben [1].

Das zentrale Element in der *Project Usus*-Perspektive ist die *Cockpit*-View. Hier wird der aktuelle Stand verschiedener im Workspace gemessener Eigenschaften des Codes dargestellt. Dazu gehören Qualitätsindikatoren wie durchschnittliche Komponentenabhängigkeit (Average Component Dependency, ACD) oder die Zahl der Methoden, die gewisse Schwellenwerte hinsichtlich der Anzahl der Statements oder der zyklomatischen Komplexität überschreiten. Ebenso angezeigt werden die Testabdeckung des letzten Laufs der *AllTests*-Suite und die Zahl der Compilerwarnungen. Die Messung erstreckt sich dabei auf alle Workspace-Projekte, die in der View *Covered Projects* ausgewählt wurden. So kann die Beobachtung der Qualitätsindikatoren auch selektiv erfolgen, statt sich auf den gesamten Workspace zu erstrecken. Durch Doppelklick auf Elemente im Cockpit gelangt man zu den *Hotspots*, einer sortierten Liste der jeweils schwerwiegendsten Überschreitungen. Beispielsweise listet

Indicator	SQI	Violations	Total
<input type="checkbox"/> Code proportions			
Cyclomatic complexity	100	0	699 methods
Average component dependency	100	17	141 classes
Class size	93.07	1	141 classes
Method length	100	0	699 methods
<input type="checkbox"/> Static analysis warnings (Yellownes)			
Compiler warnings	100	0	157 files
<input type="checkbox"/> Test coverage (Red-/Greenness)			
Test coverage	8.64	719	8324 lines
<input type="checkbox"/> Bugs			

Abb. 1: Das Usus-Cockpit

das Öffnen der Hotspots für die Metrik *Method Length* alle Methoden mit mehr als 15 Statements auf. Der Klick auf diese wiederum führt direkt an den Quellcode der jeweiligen Methode im Java-Editor. Diese Benutzerführung ermöglicht ein schnelles und systematisches Anvisieren der „neuralgischen Punkte“ für Codereviews oder Refactorings. Da alle Berechnungen inkrementell beim Speichern aktualisiert werden, wird das Ergebnis solcher Refactorings auch unmittelbar als Feedback für den Entwickler sichtbar (Abb. 1).

Checkpoints und Checkpoints History

Zu jedem Zeitpunkt, zu dem alle von Usus analysierten Parameter vorliegen (wenn also sämtliche Metriken, Compilerwarnungen und Testabdeckung gemessen wurden), wird automatisch ein *Checkpoint* registriert und auf dem Diagramm in der Checkpoints History dargestellt. So können die Auswirkungen von Codeänderungen im zeitlichen Verlauf beobachtet und das Wechselspiel der Indikatoren verfolgt werden. Beispielsweise führen Refactorings mit dem Ziel, die Methodenlänge und -komplexität zwecks besserer Lesbarkeit zu reduzieren, oft zu größeren Klassen und damit zwar zu einer Verbesserung von *Method Length* und *Cyclomatic Complexity*, aber zu einer Verschlechterung der *Class Size* – die nun in einem weiteren Schritt korrigiert

werden kann (z. B. durch Aufteilen der Klasse).

Projekteinstellungen

Einstellungen für den Editor oder Compiler, aber auch für andere integrierte Werkzeuge (wie die statische Analyse mit Checkstyle), können für jedes Projekt im Workspace individuell vorgenommen werden. Wer diese Freiheit jedoch auch tatsächlich nutzen möchte, steht vor einer aufwändigen und fehleranfälligen Konfigurationsaufgabe an den vielen *Properties*-Dialogen eines jeden einzelnen Workspace-Projekts. Usus schafft hier schnelle Abhilfe, indem es erlaubt, einen vorkonfigurierten Satz von Einstellungen direkt auf alle Projekte anzuwenden oder aber die Einstellungen eines „Master“-Projekts auf beliebige andere Projekte zu kopieren.



Leif Frenzel (andrena objects ag) ist Senior Softwareentwickler und Coach für agile Methoden. Er verfügt über langjährige Erfahrung in der Entwicklung Eclipse-basierter Software und interessiert sich besonders für die Unterstützung sauberer Coding-Praxis.



Nicole Rauch (andrena objects ag) ist Softwareentwicklerin und Team Lead mit umfangreichem Hintergrund in Compilerbau und formalen Verifikationsmethoden.



Stefan Schürle (andrena objects ag) ist Softwareentwickler mit besonderem Schwerpunkt auf der Entwicklung mit Eclipse RCP.

» Links & Literatur

[1] <http://projectusus.org>