Vorschläge zur Verbesserung der Faktor-IPS Entwicklung

von Alexander Weickmann

Hier einige Ideen, wie man die Entwicklung von Faktor-IPS meiner Meinung nach effizienter gestalten könnte (weniger Fehler, höhere Codequalität).

Spelling aktivieren

Preferences → Editors → Text Editors → Spelling → Enable Spellchecking

Jeder Entwickler sollte diese Option aktivieren. Nicht nur werden Fehler in den JavaDocs vermieden, sondern wichtiger auch Fehler in den messages – Dateien. Rot unterringelte Texte sind meist immer ein Fehler, außer handelt sich um einen Variablenname oder einen Eigenname. Dem sollte mehr Beachtung geschenkt werden.

Dieser Vorschlag zur Verbesserung der Entwicklung bedeutet keinerlei Aufwand oder Konfiguration. Es muss nur jeder Entwickler den Haken setzen und auf die roten Kringel achten :-)

Aktueller Schreibfehler in einer Faktor-IPS messages – Datei, der mit Spellchecking einfach hätte vermieden werden können:

```
configure a poliy component type, if
```

Projektspezifische Save Actions konfigurieren & aktivieren

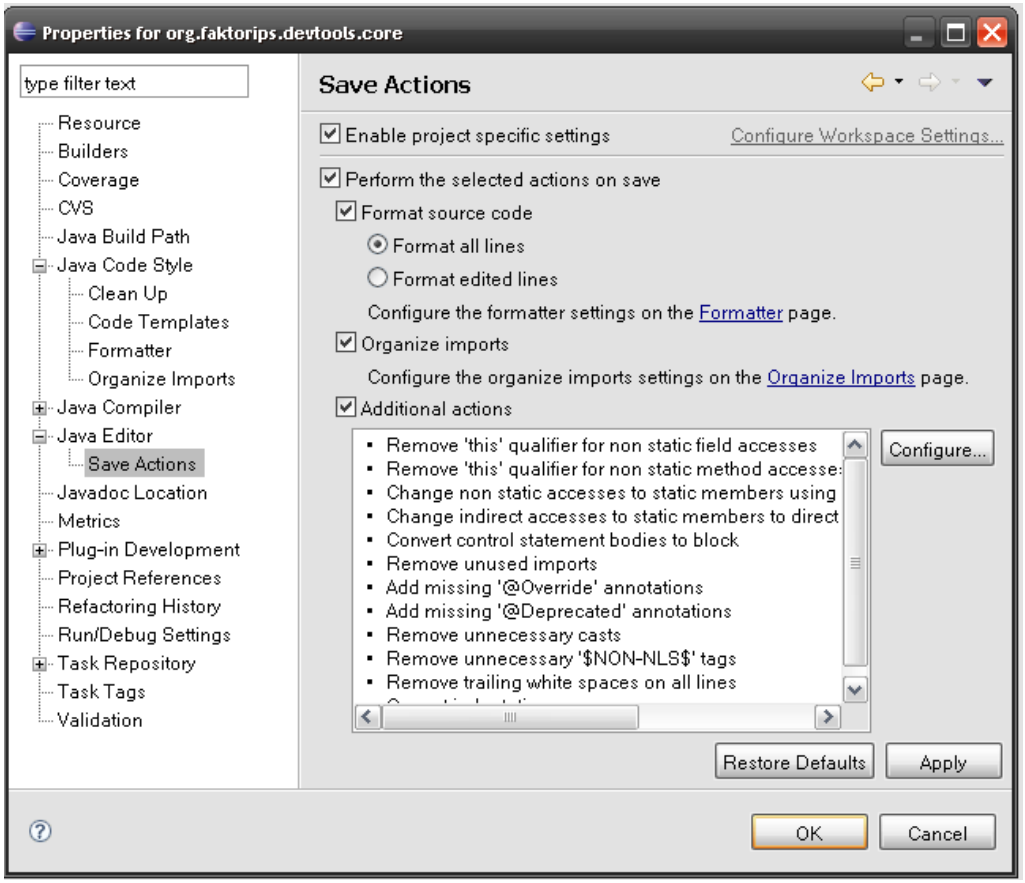
Rechtsklick auf ein Java-Projekt → Java Editor → Save Actions

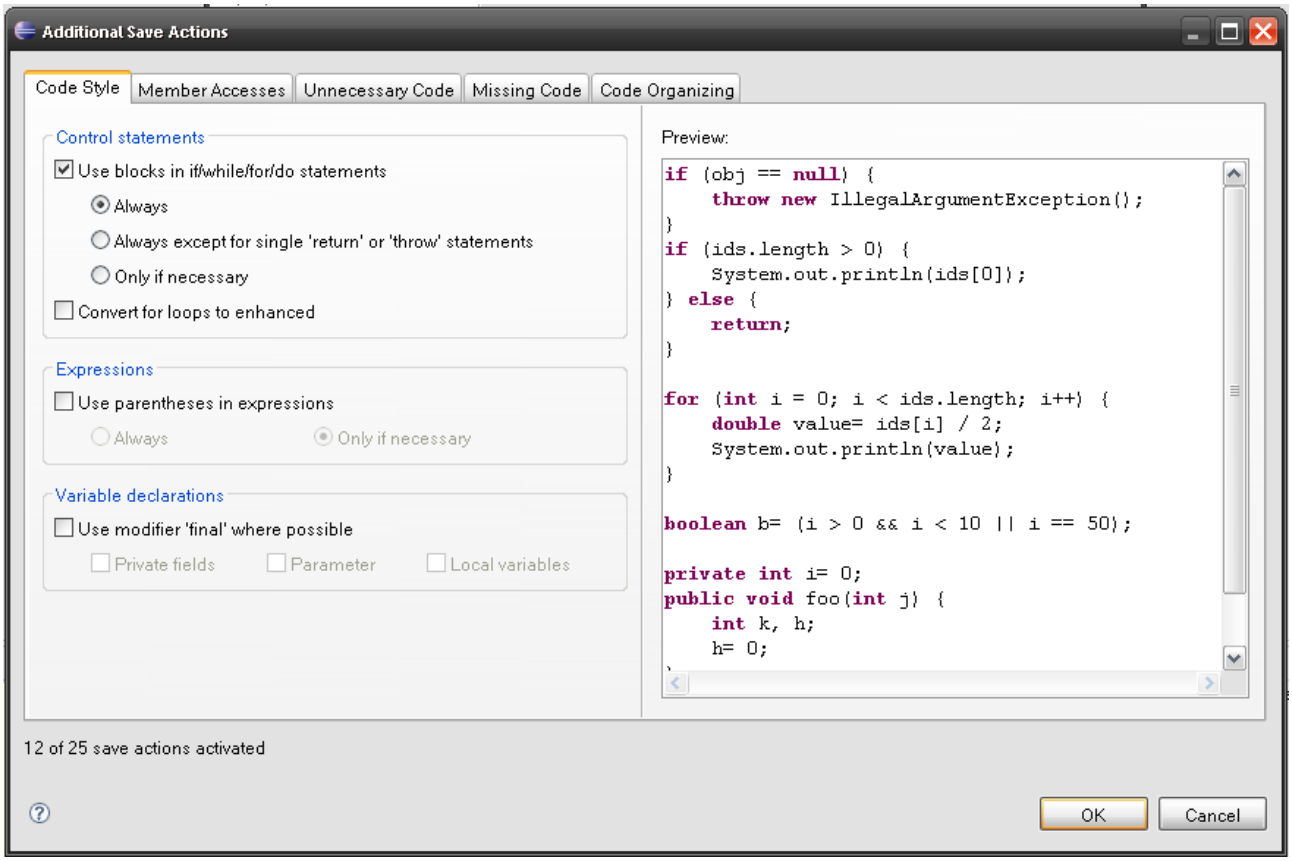
Projektspezifische Save Actions sollten für alle Projekt aktiviert und gleich konfiguriert werden. Die Einstellungen werden auch ins CVS übernommen, somit erreichen sie direkt alle Entwickler (einmaliger, geringer Konfigurationsaufwand).

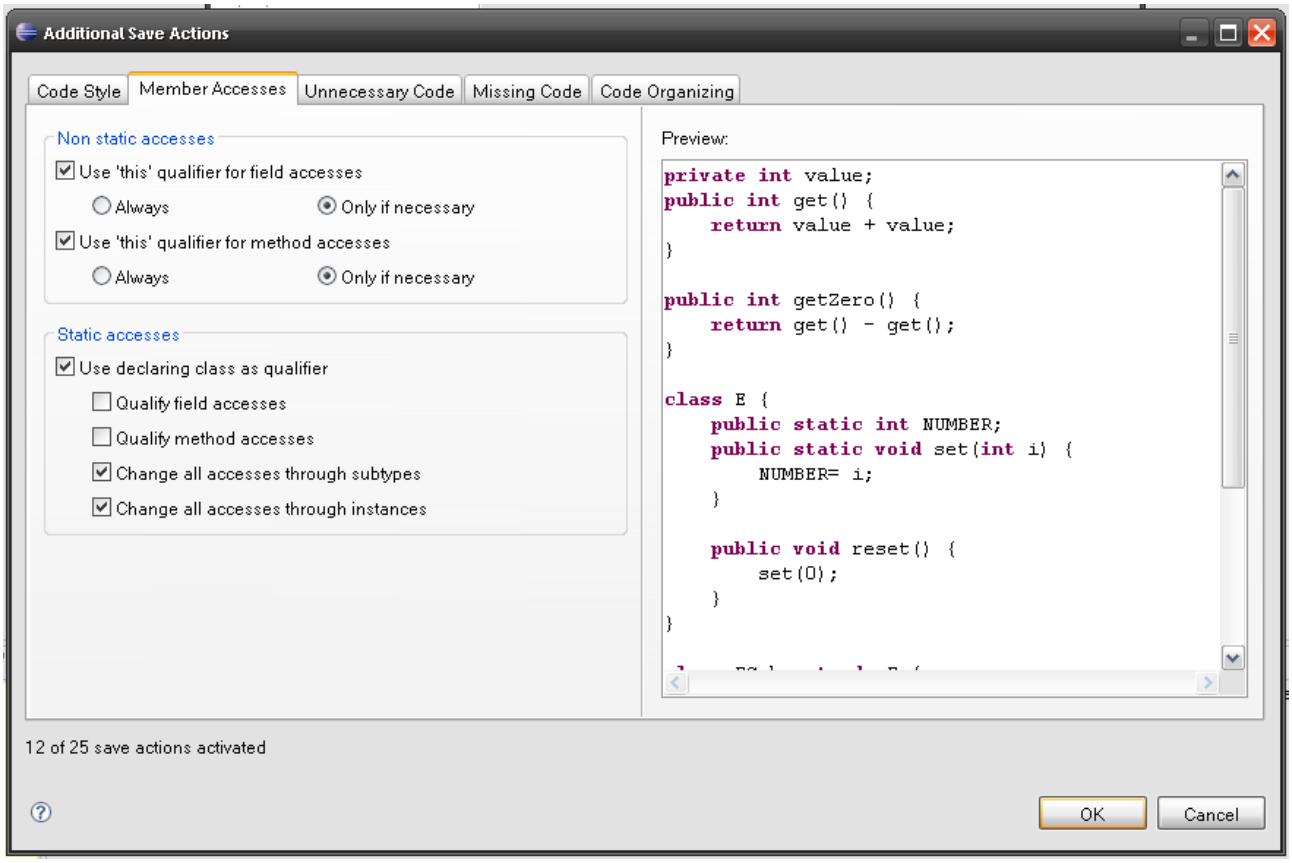
Eine Save Action ist einfach eine Aktion, die immer ausgeführt wird, wenn man eine Java-Datei abspeichert. Hier können unterschiedliche Aktionen definiert werden. Sinnvoll ist es beispielsweise, den Code automatisch formatieren zu lassen und überflüssige Imports automatisch entfernen zu lassen.

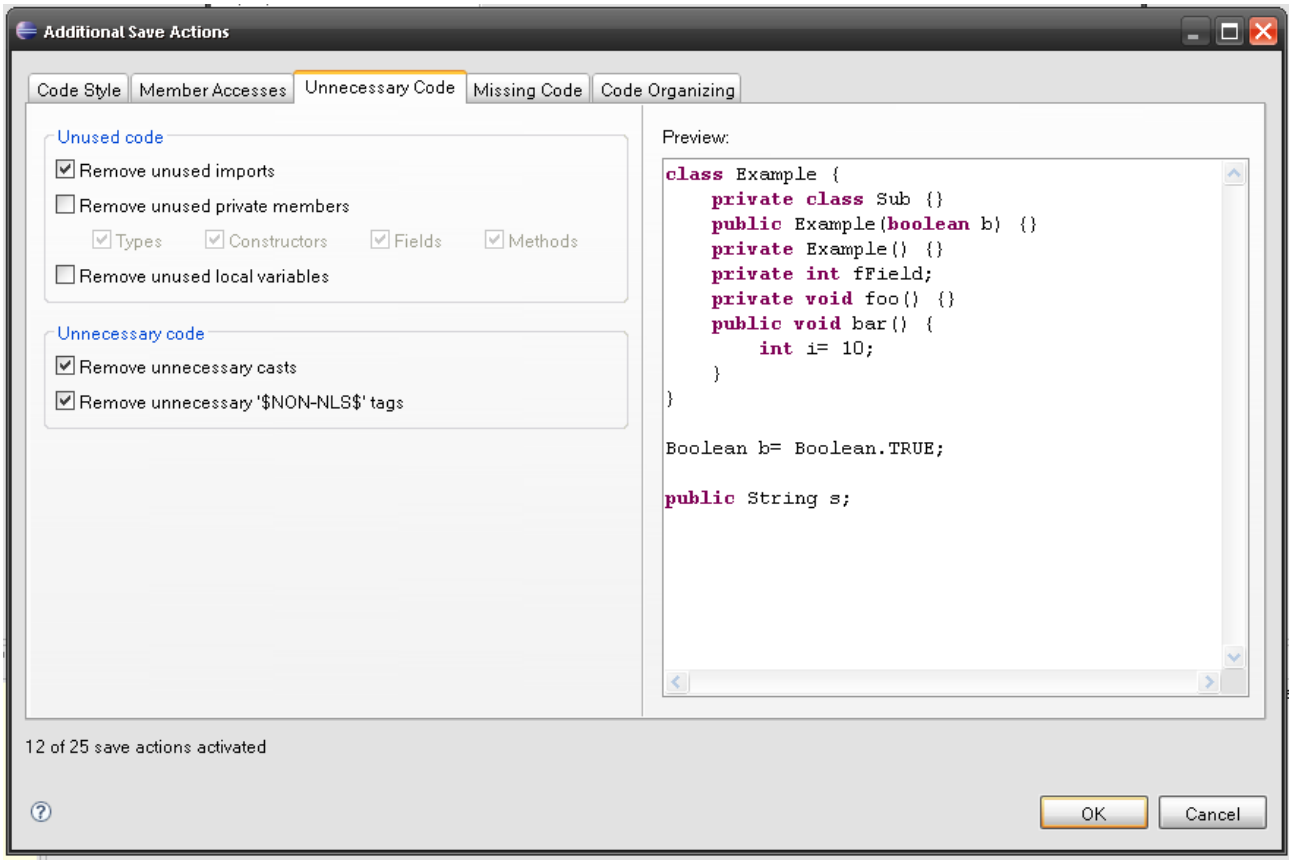
Save Actions verbessern somit „kostenlos“ die Codequalität bzw. Lesbarkeit. Außerdem sollten Formatter und Code Templates unter Java Code Style ebenfalls projektspezifisch für Faktor-IPS eingestellt werden, somit muss das nicht mehr jeder Entwickler individuell machen.

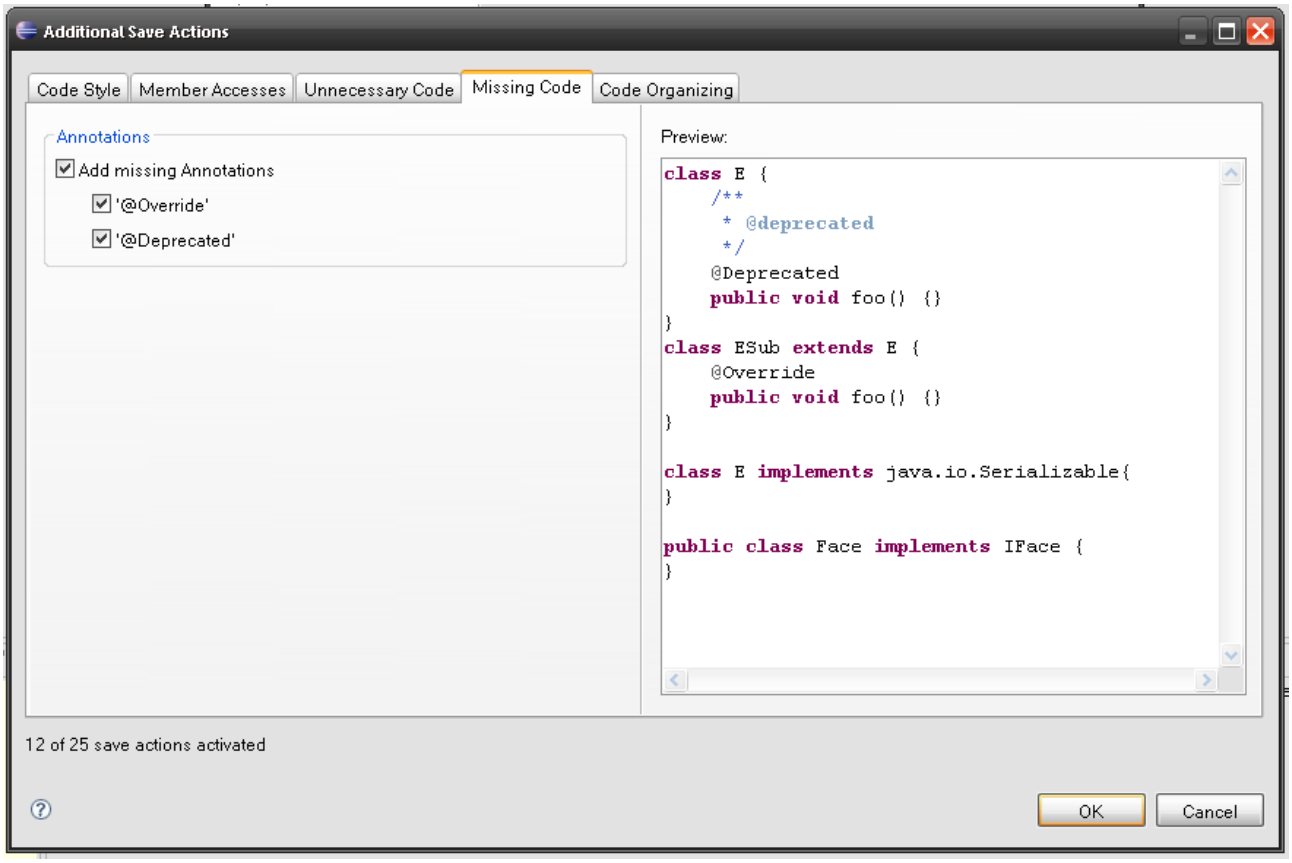
Auf den nächsten Seiten folgt ein Konfigurationsvorschlag:

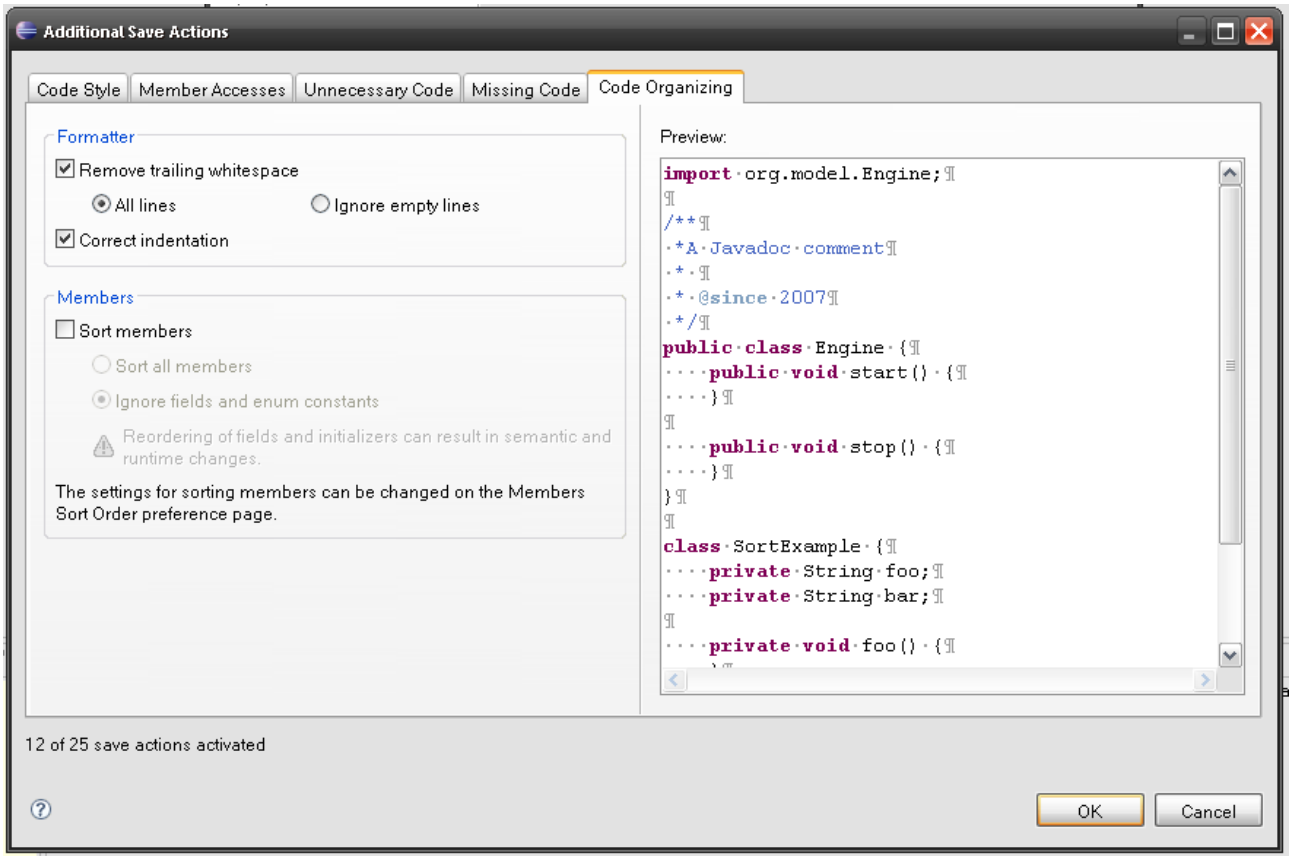

Properties for org.faktorips.devtools.core
type filter text
Resource
Builders
Coverage
CVS
Java Build Path
Java Code Style
Clean Up
Code Templates
Formatter
Organize Imports
Java Compiler
Java Editor
Save Actions
Javadoc Location
Metrics
Plug-in Development
Project References
Refactoring History
Run/Debug Settings
Task Repository
Task Tags
Validation
Save Actions
Enable project specific settings
Configure Workspace Settings...
Perform the selected actions on save
Format source code
Format all lines
Format edited lines
Configure the formatter settings on the Formatter page.
Organize imports
Configure the organize imports settings on the Organize Imports page.
Additional actions
Remove 'this' qualifier for non static field accesses
Remove 'this' qualifier for non static method accesse
Change non static accesses to static members using
Change indirect accesses to static members to direct
Convert control statement bodies to block
Remove unused imports
Add missing '@Override' annotations
Add missing '@Deprecated' annotations
Remove unnecessary casts
Remove unnecessary '$NON-NLS$' tags
Remove trailing white spaces on all lines
Configure...
Restore Defaults
Apply
OK
Cancel


Additional Save Actions
Code Style
Member Accesses
Unnecessary Code
Missing Code
Code Organizing
Control statements
Use blocks in if/while/for/do statements
Always
Always except for single 'return' or 'throw' statements
Only if necessary
Convert for loops to enhanced
Expressions
Use parentheses in expressions
Always
Only if necessary
Variable declarations
Use modifier 'final' where possible
Private fields
Parameter
Local variables
Preview:
if (obj == null) {
throw new IllegalArgumentException();
}
if (ids.length > 0) {
System.out.println(ids[0]);
} else {
return;
}
for (int i = 0; i < ids.length; i++) {
double value= ids[i] / 2;
System.out.println(value);
}
boolean b= (i > 0 && i < 10 || i == 50);
private int i= 0;
public void foo(int j) {
int k, h;
h= 0;
12 of 25 save actions activated
OK
Cancel


Additional Save Actions
Code Style
Member Accesses
Unnecessary Code
Missing Code
Code Organizing
Non static accesses
Use 'this' qualifier for field accesses
Always
Only if necessary
Use 'this' qualifier for method accesses
Always
Only if necessary
Static accesses
Use declaring class as qualifier
Qualify field accesses
Qualify method accesses
Change all accesses through subtypes
Change all accesses through instances
Preview:
private int value;
public int get() {
return value + value;
}
public int getZero() {
return get() - get();
}
class E {
public static int NUMBER;
public static void set(int i) {
NUMBER= i;
}
public void reset() {
set(0);
}
}
12 of 25 save actions activated
OK
Cancel


Additional Save Actions
Code Style
Member Accesses
Unnecessary Code
Missing Code
Code Organizing
Unused code
Remove unused imports
Remove unused private members
Types
Constructors
Fields
Methods
Remove unused local variables
Unnecessary code
Remove unnecessary casts
Remove unnecessary '$NON-NLS$' tags
Preview:
class Example {
private class Sub {}
public Example(boolean b) {}
private Example() {}
private int fField;
private void foo() {}
public void bar() {
int i= 10;
}
}
Boolean b= Boolean.TRUE;
public String s;
12 of 25 save actions activated
OK
Cancel


Additional Save Actions
Code Style
Member Accesses
Unnecessary Code
Missing Code
Code Organizing
Annotations
Add missing Annotations
'@Override'
'@Deprecated'
Preview:
class E {
/**
* @deprecated
*/
@Deprecated
public void foo() {}
}
class ESub extends E {
@Override
public void foo() {}
}
class E implements java.io.Serializable{
}
public class Face implements IFace {
}
12 of 25 save actions activated
OK
Cancel


Additional Save Actions
Code Style
Member Accesses
Unnecessary Code
Missing Code
Code Organizing
Formatter
Remove trailing whitespace
All lines
Ignore empty lines
Correct indentation
Members
Sort members
Sort all members
Ignore fields and enum constants
Reordering of fields and initializers can result in semantic and runtime changes.
The settings for sorting members can be changed on the Members Sort Order preference page.
Preview:
import org.model.Engine;
/**
* A Javadoc comment
*
* @since 2007
*/
public class Engine {
public void start() {
}
public void stop() {
}
}
class SortExample {
private String foo;
private String bar;
private void foo() {
12 of 25 save actions activated
OK
Cancel

Metriken beachten

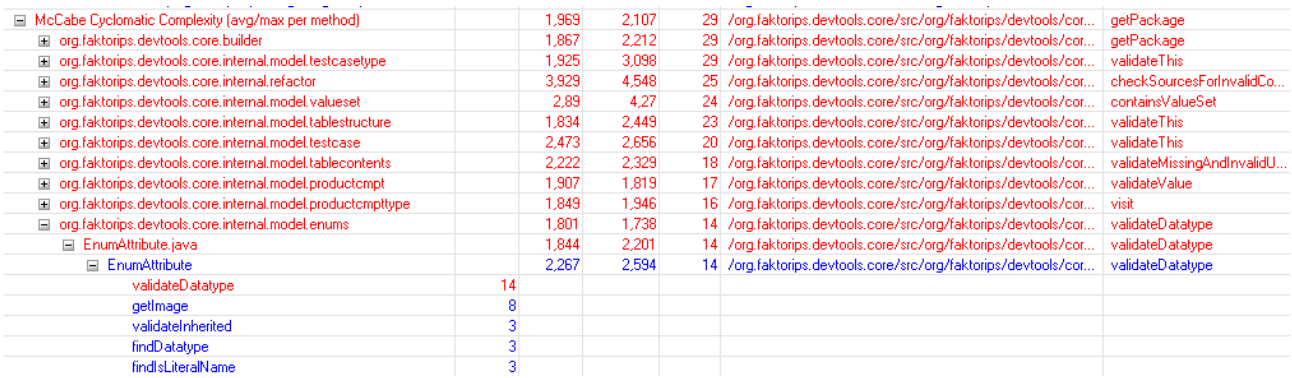
Jeder Entwickler Eclipse Metrics Plug-In installieren: metrics.sourceforge.net

Die Code-Metriken, welche von dem Plug-In zur Verfügung gestellt werden, können ähnlich zur Verbesserung der Codequalität eingesetzt werden, wie die Informationen, die das Test Coverage Plug-In EclEmma bietet.

Dies soll nicht etwas sein, wo man ständig darauf schaut. Hat man eine größere Arbeitseinheit beendet, sollte man die Metriken beachten (z.B. nach Fertigstellung der neuen Enums für diesen Code die Metriken überprüfen).

Das Plug-In hebt Codestellen rot hervor, welche höchstwahrscheinlich zu komplex sind und zumindest überdacht werden sollten. Meist tritt folgender Effekt beim Entwickler auf: „Aha, diese Methode ist angeblich zu komplex … hmm ist wirklich etwas kompliziert, teilen wir besser auf einige private Methoden auf“. Damit ist dann eine Verbesserung der Codequalität hinsichtlich Lesbarkeit und Wartbarkeit entstanden.

Beispiel:
Die Methode *validateDatatype* von *EnumAttribute.java* ist angeblich zu komplex:



McCabe Cyclomatic Complexity (avg/max per method)		1,969	2,107	29	/org.faktorips.devtools.core/src/org/faktorips/devtools/cor...	getPackage
org.faktorips.devtools.core.builder		1,867	2,212	29	/org.faktorips.devtools.core/src/org/faktorips/devtools/cor...	getPackage
org.faktorips.devtools.core.internal.model.testcasetype		1,925	3,098	29	/org.faktorips.devtools.core/src/org/faktorips/devtools/cor...	validateThis
org.faktorips.devtools.core.internal.refactor		3,929	4,548	25	/org.faktorips.devtools.core/src/org/faktorips/devtools/cor...	checkSourcesForInvalidCo...
org.faktorips.devtools.core.internal.model.valueset		2,89	4,27	24	/org.faktorips.devtools.core/src/org/faktorips/devtools/cor...	containsValueSet
org.faktorips.devtools.core.internal.model.tablestructure		1,834	2,449	23	/org.faktorips.devtools.core/src/org/faktorips/devtools/cor...	validateThis
org.faktorips.devtools.core.internal.model.testcase		2,473	2,656	20	/org.faktorips.devtools.core/src/org/faktorips/devtools/cor...	validateThis
org.faktorips.devtools.core.internal.model.tablecontents		2,222	2,329	18	/org.faktorips.devtools.core/src/org/faktorips/devtools/cor...	validateMissingAndInvalidU...
org.faktorips.devtools.core.internal.model.productcmpt		1,907	1,819	17	/org.faktorips.devtools.core/src/org/faktorips/devtools/cor...	validateValue
org.faktorips.devtools.core.internal.model.productcmpttype		1,849	1,946	16	/org.faktorips.devtools.core/src/org/faktorips/devtools/cor...	visit
org.faktorips.devtools.core.internal.model.enums		1,801	1,738	14	/org.faktorips.devtools.core/src/org/faktorips/devtools/cor...	validateDatatype
EnumAttribute.java		1,844	2,201	14	/org.faktorips.devtools.core/src/org/faktorips/devtools/cor...	validateDatatype
EnumAttribute		2,267	2,594	14	/org.faktorips.devtools.core/src/org/faktorips/devtools/cor...	validateDatatype
validateDatatype	14					
getImage	8					
validateInherited	3					
findDatatype	3					
findIsLiteralName	3					

Werfen wir einen Blick auf den Quellcode:

```
/** Validates the <code>datatype</code> property. */
    private void validateDatatype(MessageList list, IIpsProject
ipsProject) throws CoreException {
        String text;
        Message validationMessage;
        IEnumType enumType = getEnumType();
        Datatype ipsDatatype = getIpsProject().findDatatype(datatype);

        // The datatype must be specified.
        if (datatype.equals("")) { //$NON-NLS-1$
            text = Messages.EnumAttribute_DatatypeMissing;
            validationMessage = new
Message(MSGCODE_ENUM_ATTRIBUTE_DATATYPE_MISSING, text, Message.ERROR,
this,
                    PROPERTY_DATATYPE);
            list.add(validationMessage);
            return;
        }

        // The datatype must exist.
```

```
        if (ipsDatatype == null) {
            text = NLS.bind(Messages.EnumAttribute_DatatypeDoesNotExist,
datatype);
            validationMessage = new
Message(MSGCODE_ENUM_ATTRIBUTE_DATATYPE_DOES_NOT_EXIST, text,
Message.ERROR, this,
                    PROPERTY_DATATYPE);
            list.add(validationMessage);
            return;
        }

        // The datatype may not be primitive.
        if (ipsDatatype.isPrimitive()) {
            text = NLS.bind(Messages.EnumAttribute_DatatypeIsPrimitive,
datatype);
            validationMessage = new
Message(MSGCODE_ENUM_ATTRIBUTE_DATATYPE_IS_PRIMITIVE, text,
Message.ERROR, this,
                    PROPERTY_DATATYPE);
            list.add(validationMessage);
            return;
        }

        // The datatype may not be void.
        if (ipsDatatype.isVoid()) {
            text = Messages.EnumAttribute_DatatypeIsVoid;
            validationMessage = new
Message(MSGCODE_ENUM_ATTRIBUTE_DATATYPE_IS_VOID, text, Message.ERROR,
this,
                    PROPERTY_DATATYPE);
            list.add(validationMessage);
            return;
        }

        // The datatype may not be abstract.
        if (ipsDatatype.isAbstract()) {
            text = NLS.bind(Messages.EnumAttribute_DatatypeIsAbstract,
datatype);
            validationMessage = new
Message(MSGCODE_ENUM_ATTRIBUTE_DATATYPE_IS_ABSTRACT, text, Message.ERROR,
this,
                    PROPERTY_DATATYPE);
            list.add(validationMessage);
            return;
        }

        // The datatype must not be the enum type that contains this enum
attribute (or subclasses
        // of it).
        if (ipsDatatype instanceof EnumTypeDatatypeAdapter) {
            EnumTypeDatatypeAdapter adaptedEnumType =
(EnumTypeDatatypeAdapter)ipsDatatype;
            List<IEnumType> subEnumTypes =
enumType.findAllSubEnumTypes(ipsProject);
            if (adaptedEnumType.getEnumType().equals(enumType) ||
subEnumTypes.contains(adaptedEnumType.getEnumType())) {
                text =
Messages.EnumAttribute_DatatypeIsContainingEnumTypeOrSubclass;;
```

```
                validationMessage = new
Message(MSGCODE_ENUM_ATTRIBUTE_DATATYPE_IS_CONTAINING_ENUM_TYPE_OR_SUBCL
ASS,
                        text, Message.ERROR, this, PROPERTY_DATATYPE);
                list.add(validationMessage);
                return;
            }

        }

        // If this enum attribute is marked to be used as literal name
the datatype must be String.
        if (literalName) {
            if (!
(ipsDatatype.getName().equals(Datatype.STRING.getName()))) {
                text =
Messages.EnumAttribute_LiteralNameNotOfDatatypeString;
                validationMessage = new
Message(MSGCODE_ENUM_ATTRIBUTE_LITERAL_NAME_NOT_OF_DATATYPE_STRING, text,
                        Message.ERROR, this, PROPERTY_DATATYPE);
                list.add(validationMessage);
                return;
            }
        }

        /*
         * The datatype may not be an enum that does not contain values
if the enum type this enum
         * attribute belongs to does contain values.
         */
        if (ipsDatatype instanceof EnumTypeDatatypeAdapter) {
            EnumTypeDatatypeAdapter enumDatatypeAdapter =
(EnumTypeDatatypeAdapter)ipsDatatype;
            IEnumType enumDatatype = enumDatatypeAdapter.getEnumType();
            if (enumType.isContainingValues() && !
(enumDatatype.isContainingValues())) {
                text =
NLS.bind(Messages.EnumAttribute_EnumDatatypeDoesNotContainValuesButParen
tEnumTypeDoes,
                        enumDatatype.getQualifiedName());
                validationMessage = new Message(
                        MSGCODE_ENUM_ATTRIBUTE_ENUM_DATATYPE_DOES_NOT_CO
NTAIN_VALUES_BUT_PARENT_ENUM_TYPE_DOES, text,
                        Message.ERROR, this, PROPERTY_DATATYPE);
                list.add(validationMessage);
                return;
            }
        }
    }
```

Diese Methode ist viel zu lang, daher ist die Lesbarkeit nicht gut. Eine Verbesserung wäre es, die einzelnen Aspekte in eigene private Methoden auszulagern.

Eine andere Metrik ist z.B. die Anzahl Parameter von Methoden oder deren Verschachtelungstiefe. Zur Konfiguration muss lediglich ein boolesches Flag in den .project Dateien gesetzt werden.