Wissenstransfer durch leichtgewichtige Reviews

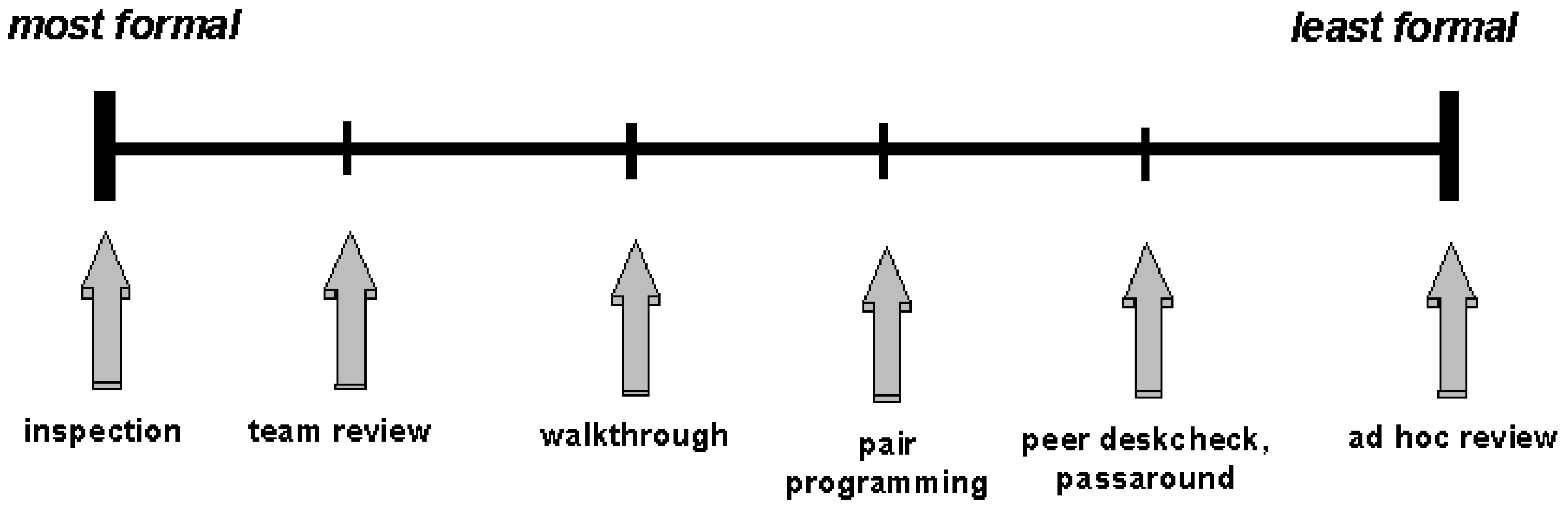
Dr. Elmar Juergens

CQSE
Continuous Quality in Software Engineering

Definition Review

Ein Review ist eine manuelle Untersuchung eines Artefakts mit dem Ziel, Probleme zu erkennen und zu beheben.

Peer-Review Arten



http://en.wikipedia.org/wiki/File:Pair_programming_1.jpg

Empirical Study

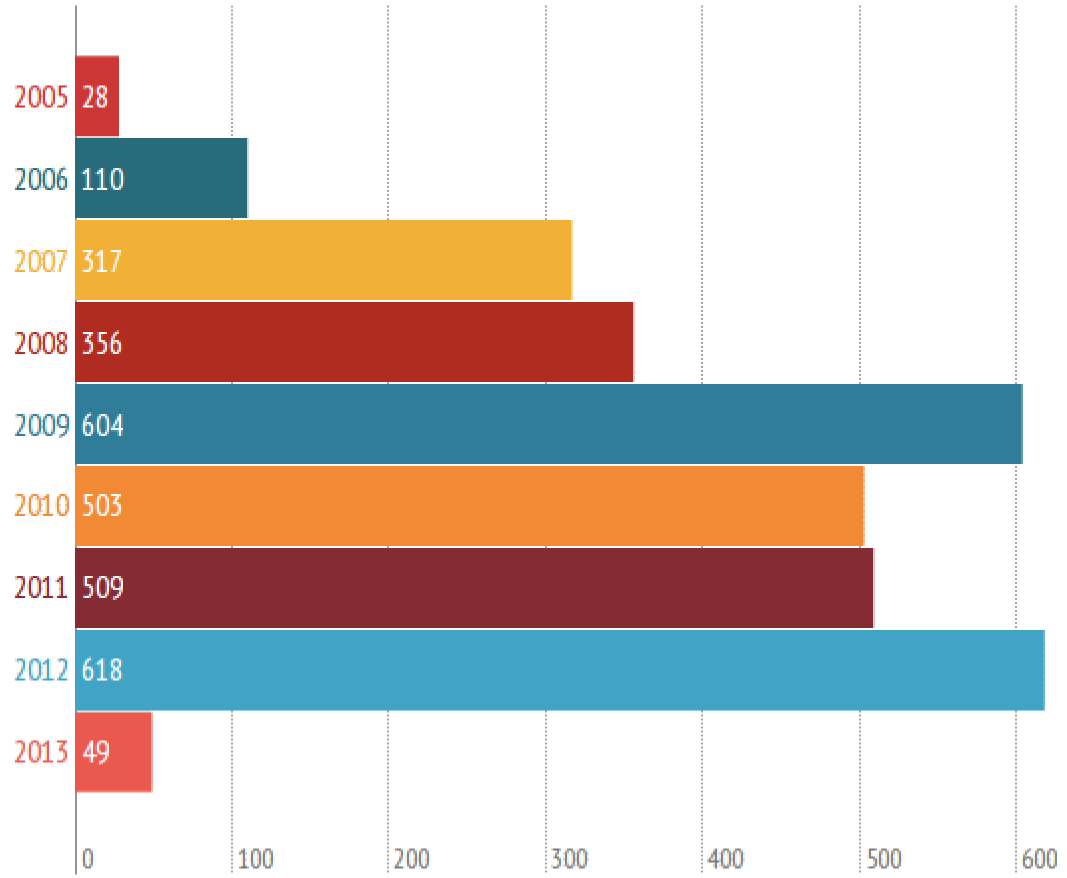
The Impact of Code Review Coverage and Code Review Participation on Software Quality, *Shane McIntosh, Yasutaka Kamei, Bram Adams, Ahmed Hassan*, MSR 2014

RQ1: Does review coverage impact post release defect counts? YES

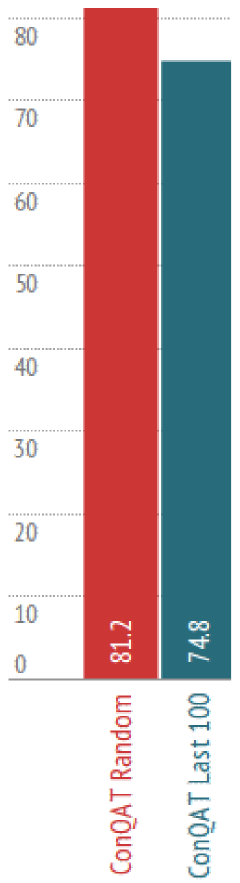
RQ2: Does review thoroughness impact post release defect counts? YES

Components with low review participation: Up to 5 more post release defects *per component*!

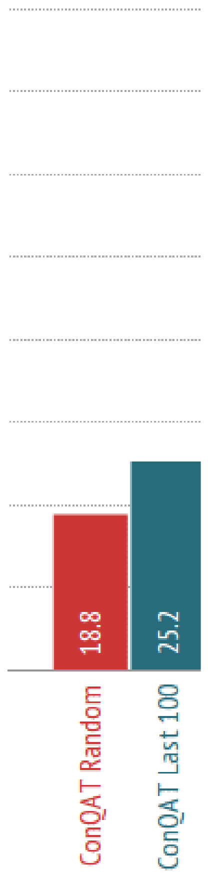
Issues per Year in ConQAT



Maintainability



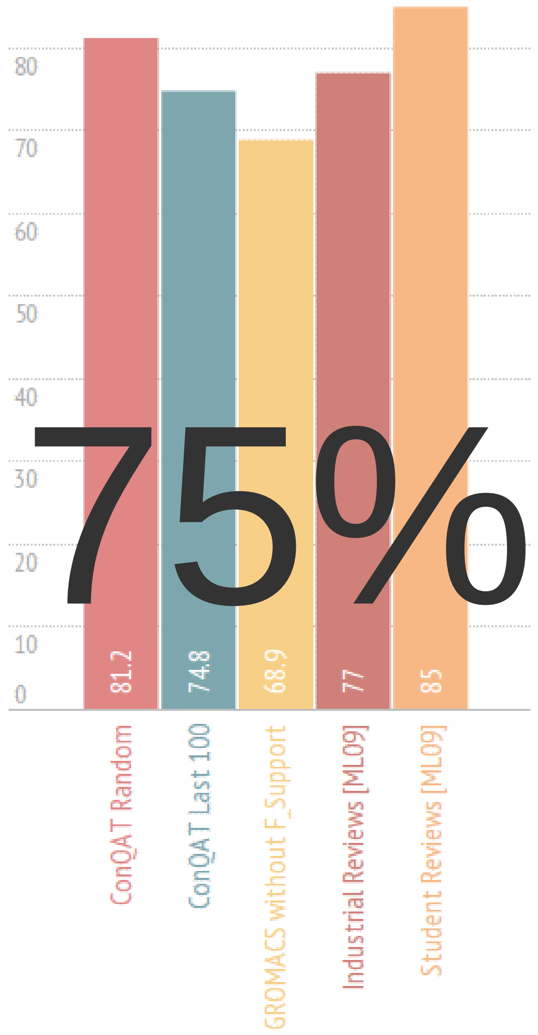
Functional

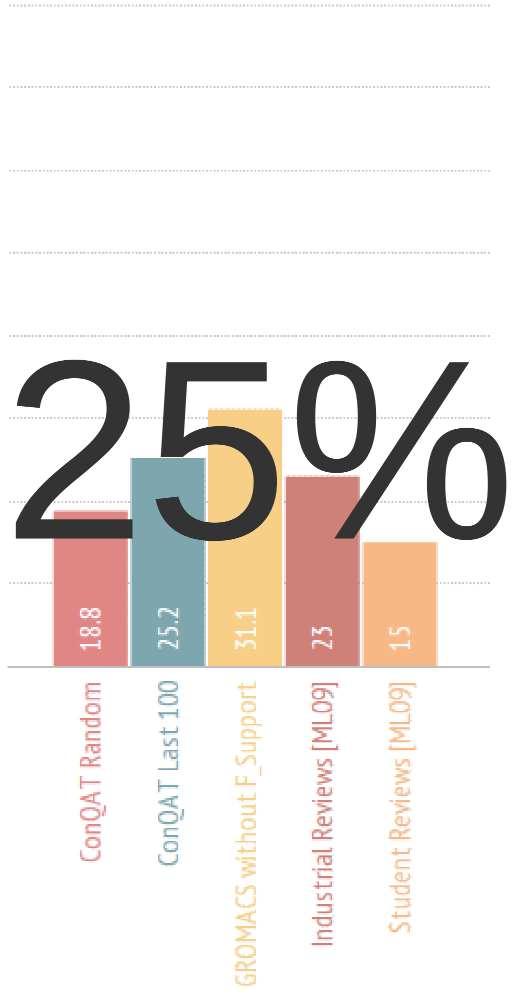


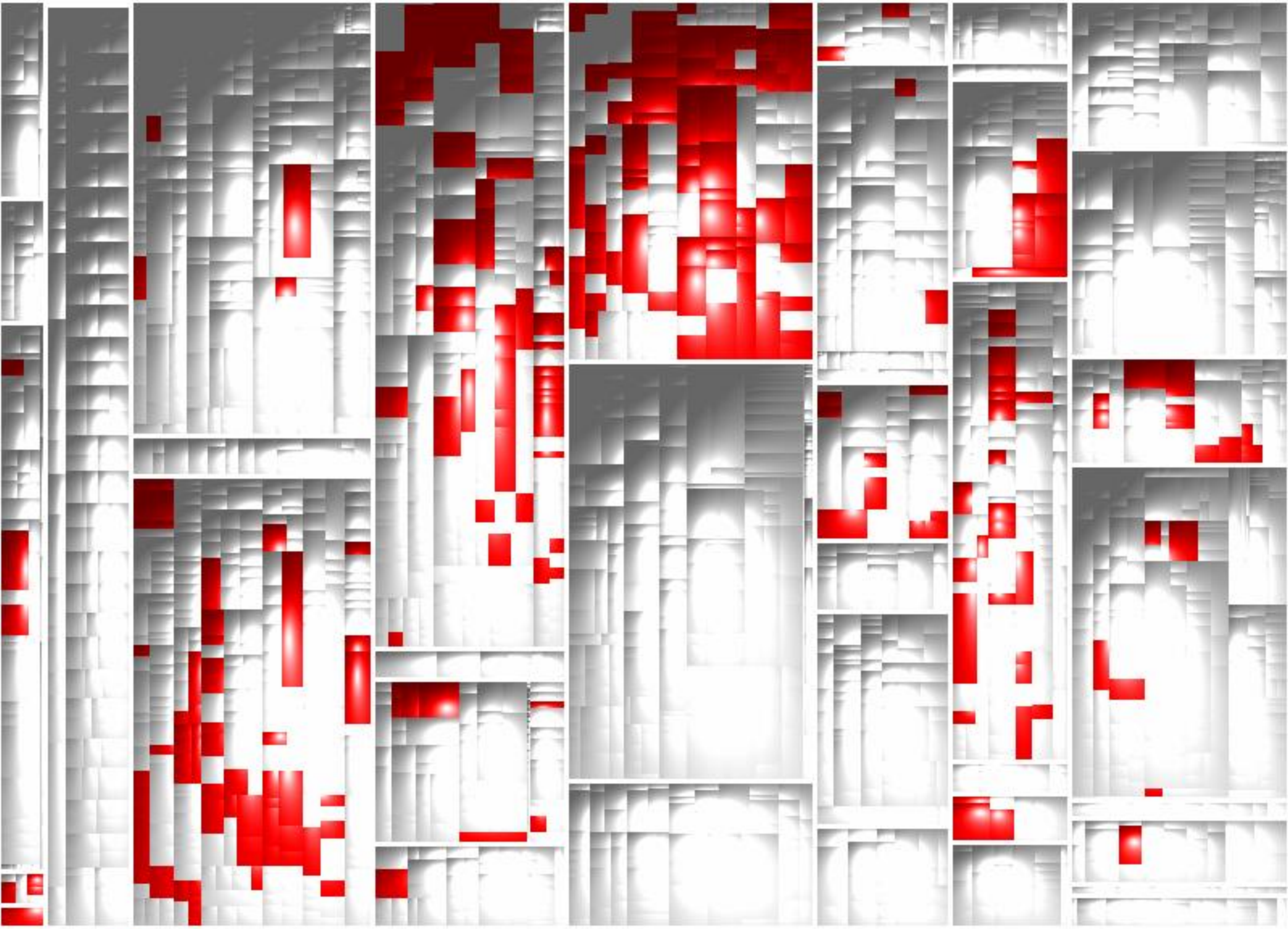
Modern Code Reviews in Open-Source Projects: Which Problems Do They Find/Fix?, Beller, Juergens, MSR 2014

Maintainability

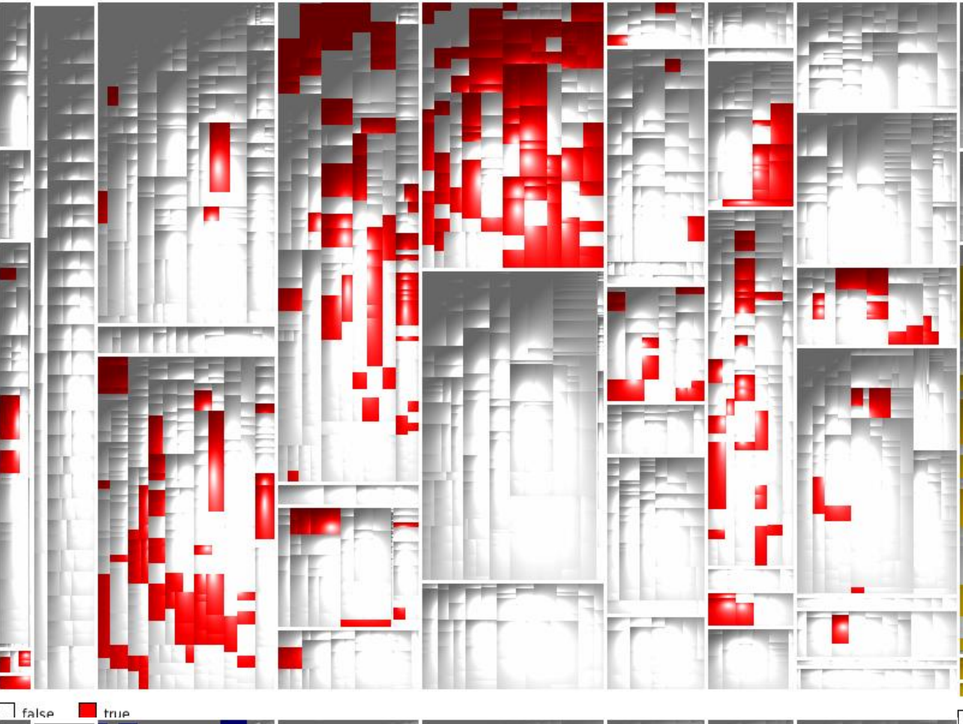
Functional

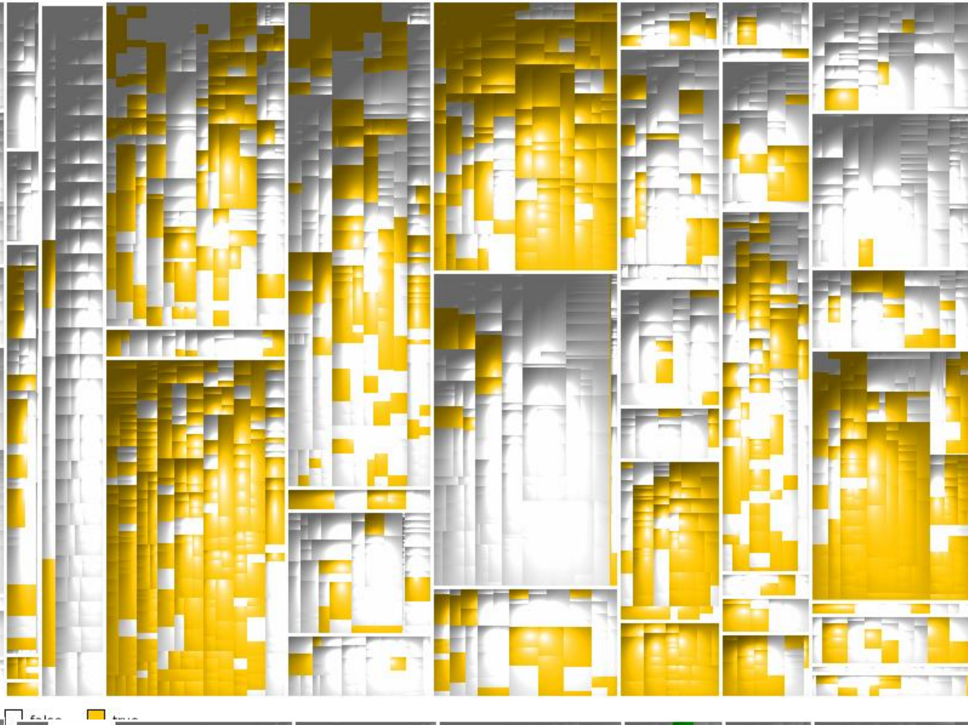


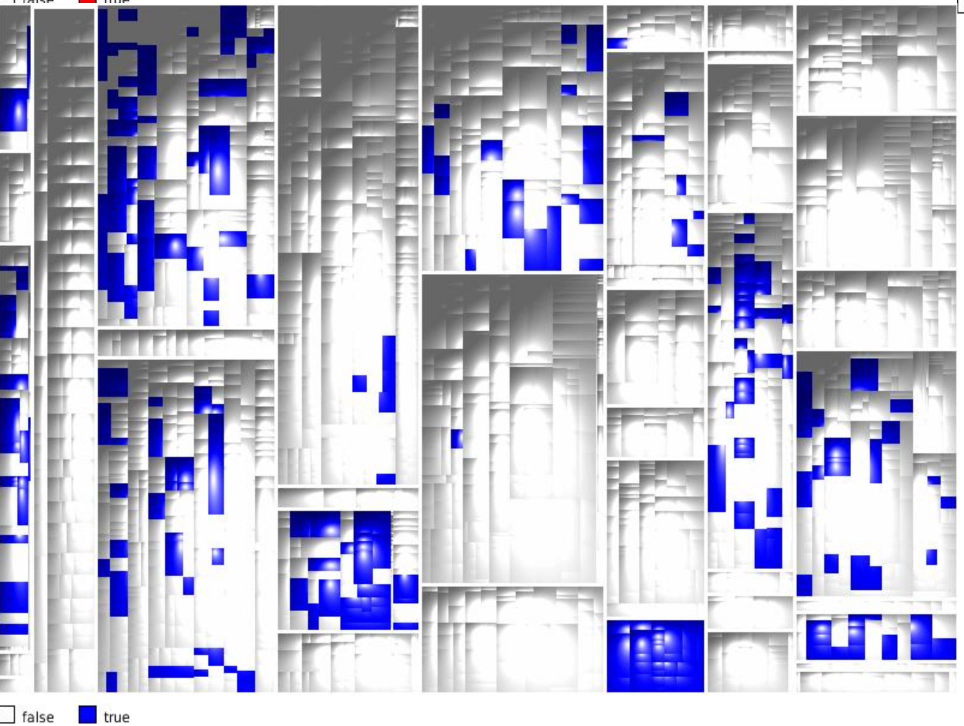


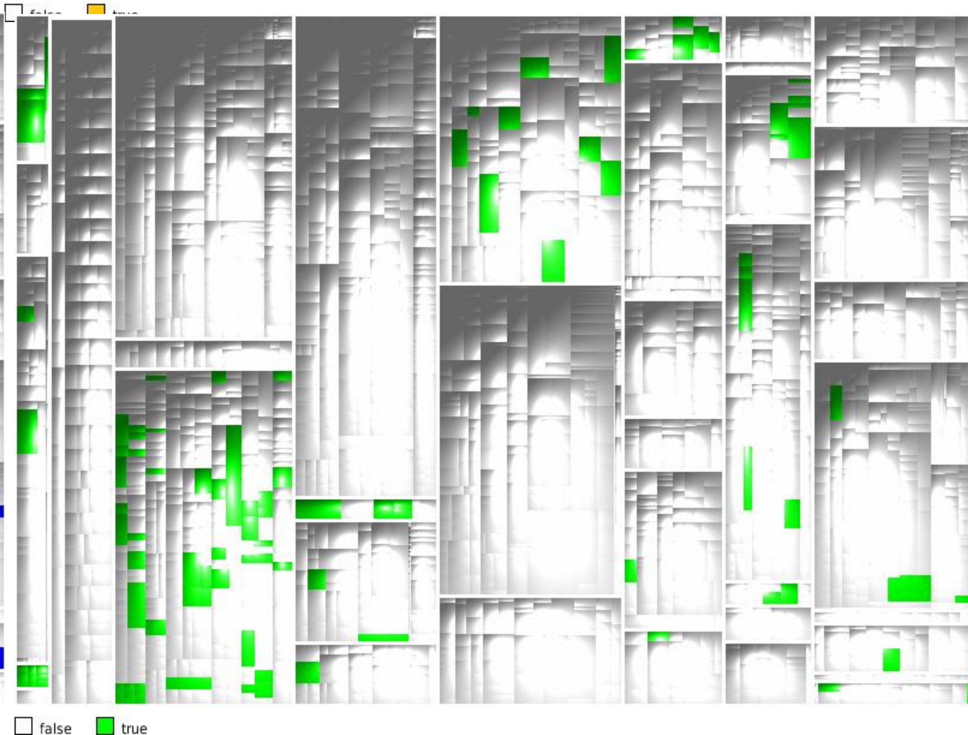


□ false 🟥 true

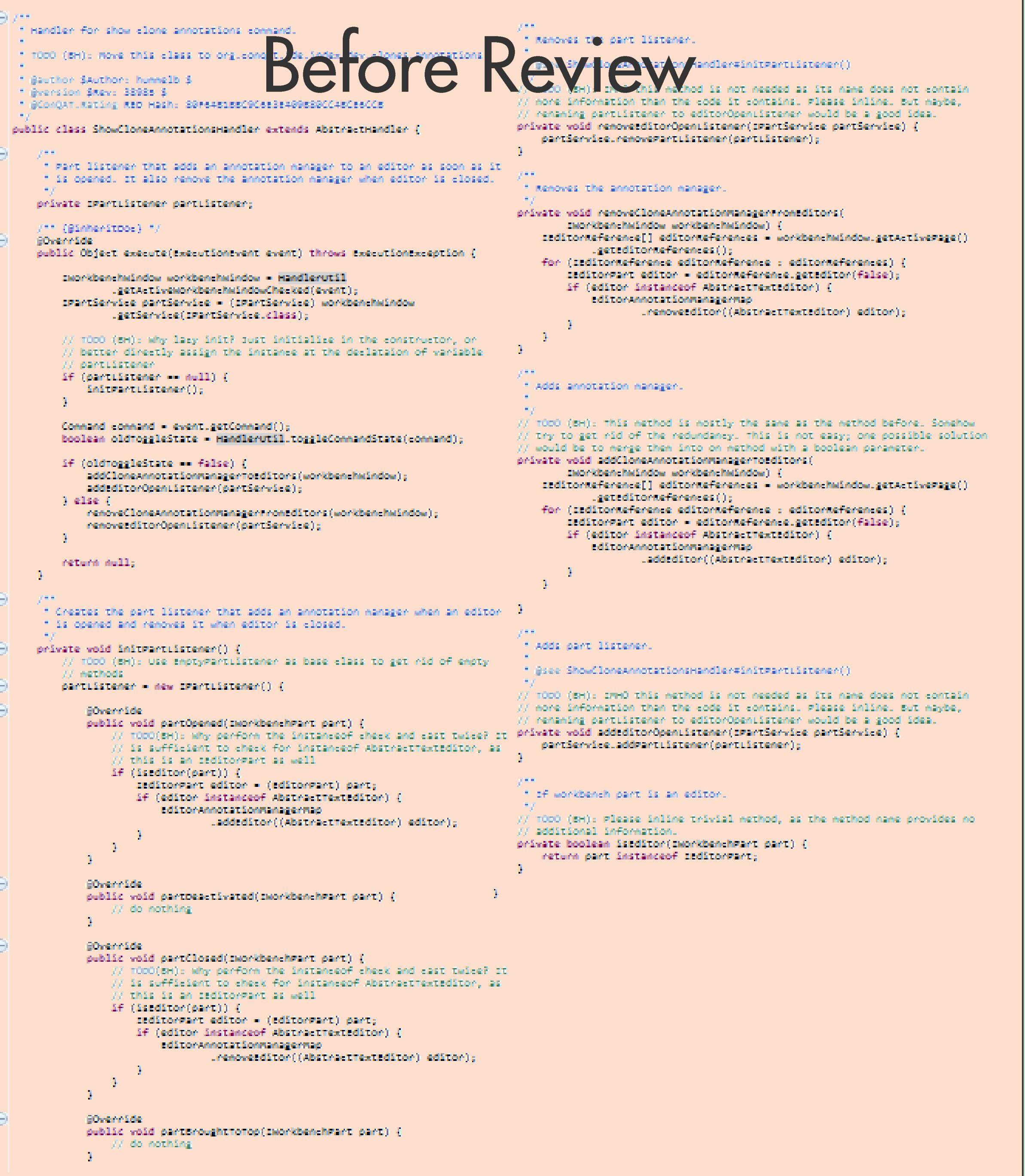

false
true


false
true


false
true


false
true

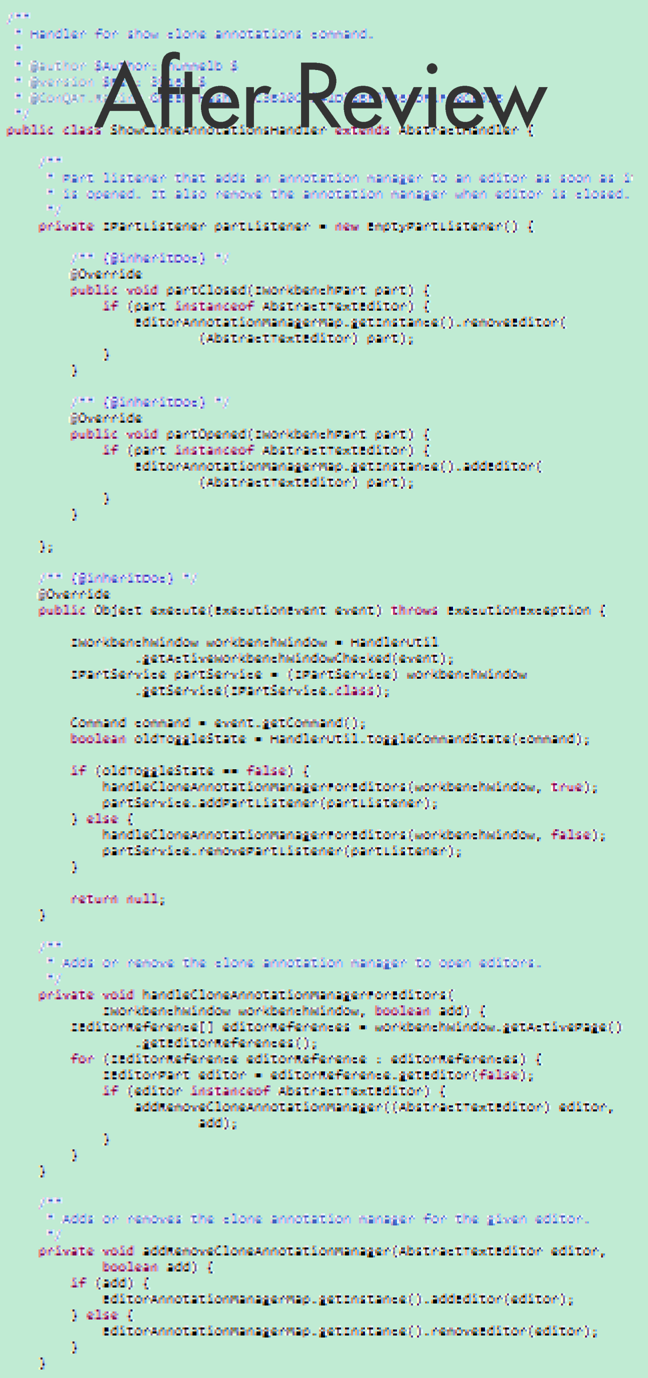
Before Review



```java
/**
 * Handler for show clone annotations command.
 *
 * TODO (BH): Move this class to org.conqat.ide.index.dev.clones.annotations
 *
 * @author $Author: hummelb $
 * @version $Rev: 33885 $
 * @ConQAT.Rating RED Hash: [illegible]
 */
public class ShowCloneAnnotationsHandler extends AbstractHandler {

    /**
     * Part listener that adds an annotation manager to an editor as soon as it
     * is opened. It also remove the annotation manager when editor is closed.
     */
    private IPartListener partListener;

    /** {@inheritDoc} */
    @Override
    public Object execute(ExecutionEvent event) throws ExecutionException {

        IWorkbenchWindow workbenchWindow = HandlerUtil
                .getActiveWorkbenchWindowChecked(event);
        IPartService partService = (IPartService) workbenchWindow
                .getService(IPartService.class);

        // TODO (BH): Why lazy init? Just initialise in the constructor, or
        // better directly assign the instance at the declatation of variable
        // partListener
        if (partListener == null) {
            initPartListener();
        }

        Command command = event.getCommand();
        boolean oldToggleState = HandlerUtil.toggleCommandState(command);

        if (oldToggleState == false) {
            addCloneAnnotationManagerToEditors(workbenchWindow);
            addEditorOpenListener(partService);
        } else {
            removeCloneAnnotationManagerFromEditors(workbenchWindow);
            removeEditorOpenListener(partService);
        }

        return null;
    }

    /**
     * Creates the part listener that adds an annotation manager when an editor
     * is opened and removes it when editor is closed.
     */
    private void initPartListener() {
        // TODO (BH): Use EmptyPartListener as base class to get rid of empty
        // methods
        partListener = new IPartListener() {

            @Override
            public void partOpened(IWorkbenchPart part) {
                // TODO(BH): Why perform the instanceof check and cast twice? It
                // is sufficient to check for instanceof AbstractTextEditor, as
                // this is an IEditorPart as well
                if (isEditor(part)) {
                    IEditorPart editor = (IEditorPart) part;
                    if (editor instanceof AbstractTextEditor) {
                        editorAnnotationManagerMap
                                .addEditor((AbstractTextEditor) editor);
                    }
                }
            }

            @Override
            public void partDeactivated(IWorkbenchPart part) {
                // do nothing
            }

            @Override
            public void partClosed(IWorkbenchPart part) {
                // TODO(BH): Why perform the instanceof check and cast twice? It
                // is sufficient to check for instanceof AbstractTextEditor, as
                // this is an IEditorPart as well
                if (isEditor(part)) {
                    IEditorPart editor = (IEditorPart) part;
                    if (editor instanceof AbstractTextEditor) {
                        editorAnnotationManagerMap
                                .removeEditor((AbstractTextEditor) editor);
                    }
                }
            }

            @Override
            public void partBroughtToTop(IWorkbenchPart part) {
                // do nothing
            }
```

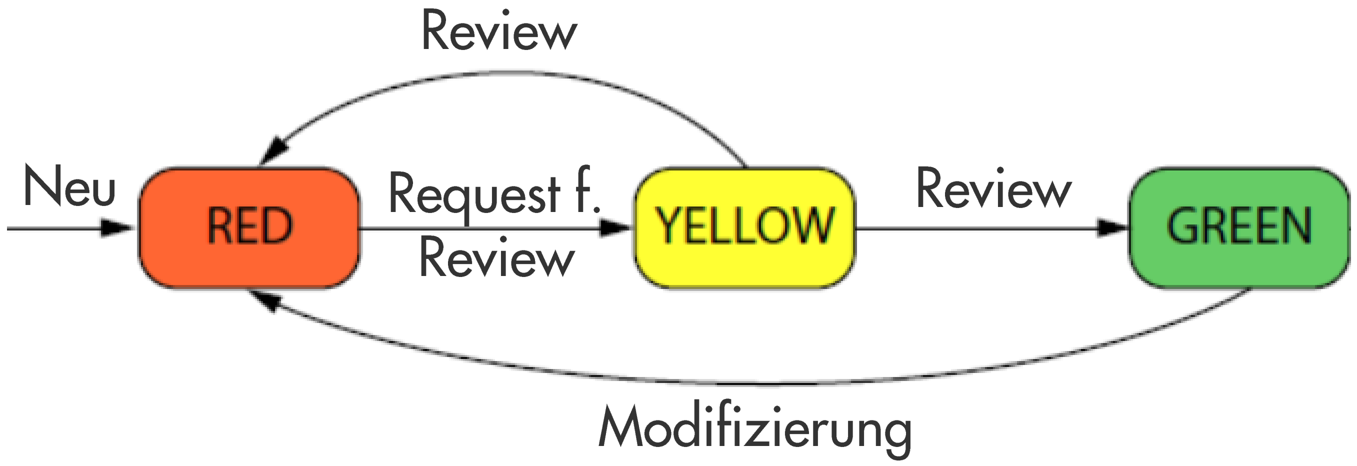
```java
    /**
     * Removes the part listener.
     *
     * @see ShowCloneAnnotationsHandler#initPartListener()
     */
    // TODO (BH): IMHO this method is not needed as its name does not contain
    // more information than the code it contains. Please inline. But maybe,
    // renaming partListener to editorOpenListener would be a good idea.
    private void removeEditorOpenListener(IPartService partService) {
        partService.removePartListener(partListener);
    }

    /**
     * Removes the annotation manager.
     */
    private void removeCloneAnnotationManagerFromEditors(
            IWorkbenchWindow workbenchWindow) {
        IEditorReference[] editorReferences = workbenchWindow.getActivePage()
                .getEditorReferences();
        for (IEditorReference editorReference : editorReferences) {
            IEditorPart editor = editorReference.getEditor(false);
            if (editor instanceof AbstractTextEditor) {
                editorAnnotationManagerMap
                        .removeEditor((AbstractTextEditor) editor);
            }
        }
    }

    /**
     * Adds annotation manager.
     *
     */
    // TODO (BH): This method is mostly the same as the method before. Somehow
    // try to get rid of the redundancy. This is not easy; one possible solution
    // would be to merge them into on method with a boolean parameter.
    private void addCloneAnnotationManagerToEditors(
            IWorkbenchWindow workbenchWindow) {
        IEditorReference[] editorReferences = workbenchWindow.getActivePage()
                .getEditorReferences();
        for (IEditorReference editorReference : editorReferences) {
            IEditorPart editor = editorReference.getEditor(false);
            if (editor instanceof AbstractTextEditor) {
                editorAnnotationManagerMap
                        .addEditor((AbstractTextEditor) editor);
            }
        }

    }

    /**
     * Adds part listener.
     *
     * @see ShowCloneAnnotationsHandler#initPartListener()
     */
    // TODO (BH): IMHO this method is not needed as its name does not contain
    // more information than the code it contains. Please inline. But maybe,
    // renaming partListener to editorOpenListener would be a good idea.
    private void addEditorOpenListener(IPartService partService) {
        partService.addPartListener(partListener);
    }

    /**
     * If workbench part is an editor.
     */
    // TODO (BH): Please inline trivial method, as the method name provides no
    // additional information.
    private boolean isEditor(IWorkbenchPart part) {
        return part instanceof IEditorPart;
    }

}
```

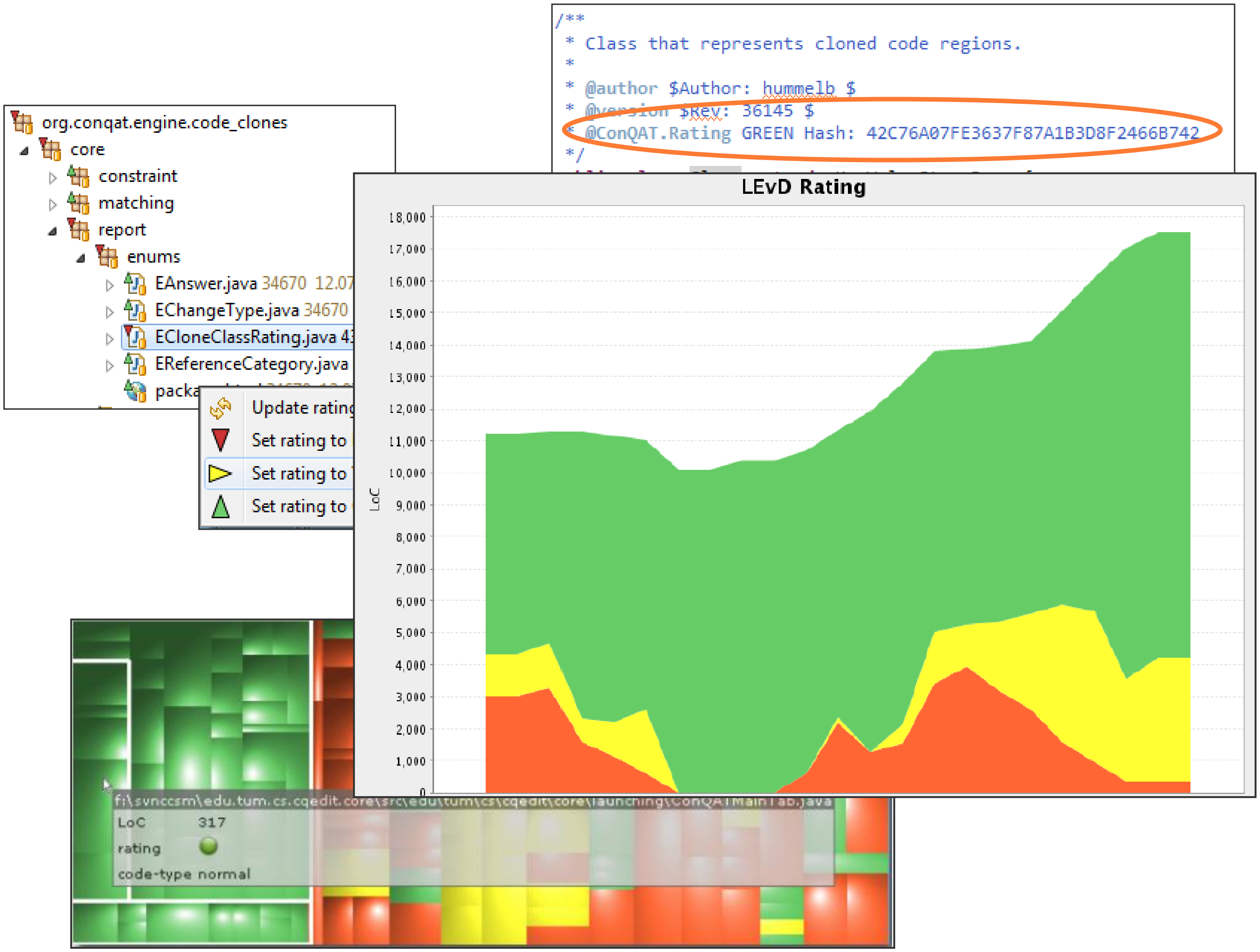
After Review



```java
/**
 * Handler for show clone annotations command.
 *
 * @author $Author: hummelb $
 * @version $Rev: [illegible] $
 * @ConQAT.Rating GREEN Hash: [illegible]
 */
public class ShowCloneAnnotationsHandler extends AbstractHandler {

    /**
     * Part listener that adds an annotation manager to an editor as soon as it
     * is opened. It also remove the annotation manager when editor is closed.
     */
    private IPartListener partListener = new EmptyPartListener() {

        /** {@inheritDoc} */
        @Override
        public void partClosed(IWorkbenchPart part) {
            if (part instanceof AbstractTextEditor) {
                EditorAnnotationManagerMap.getInstance().removeEditor(
                        (AbstractTextEditor) part);
            }
        }

        /** {@inheritDoc} */
        @Override
        public void partOpened(IWorkbenchPart part) {
            if (part instanceof AbstractTextEditor) {
                EditorAnnotationManagerMap.getInstance().addEditor(
                        (AbstractTextEditor) part);
            }
        }

    };

    /** {@inheritDoc} */
    @Override
    public Object execute(ExecutionEvent event) throws ExecutionException {

        IWorkbenchWindow workbenchWindow = HandlerUtil
                .getActiveWorkbenchWindowChecked(event);
        IPartService partService = (IPartService) workbenchWindow
                .getService(IPartService.class);

        Command command = event.getCommand();
        boolean oldToggleState = HandlerUtil.toggleCommandState(command);

        if (oldToggleState == false) {
            handleCloneAnnotationManagerForEditors(workbenchWindow, true);
            partService.addPartListener(partListener);
        } else {
            handleCloneAnnotationManagerForEditors(workbenchWindow, false);
            partService.removePartListener(partListener);
        }

        return null;
    }

    /**
     * Adds or remove the clone annotation manager to open editors.
     */
    private void handleCloneAnnotationManagerForEditors(
            IWorkbenchWindow workbenchWindow, boolean add) {
        IEditorReference[] editorReferences = workbenchWindow.getActivePage()
                .getEditorReferences();
        for (IEditorReference editorReference : editorReferences) {
            IEditorPart editor = editorReference.getEditor(false);
            if (editor instanceof AbstractTextEditor) {
                addRemoveCloneAnnotationManager((AbstractTextEditor) editor,
                        add);
            }
        }
    }

    /**
     * Adds or removes the clone annotation manager for the given editor.
     */
    private void addRemoveCloneAnnotationManager(AbstractTextEditor editor,
            boolean add) {
        if (add) {
            EditorAnnotationManagerMap.getInstance().addEditor(editor);
        } else {
            EditorAnnotationManagerMap.getInstance().removeEditor(editor);
        }
    }
```

http://en.wikipedia.org/wiki/File:Pair_programming_1.jpg

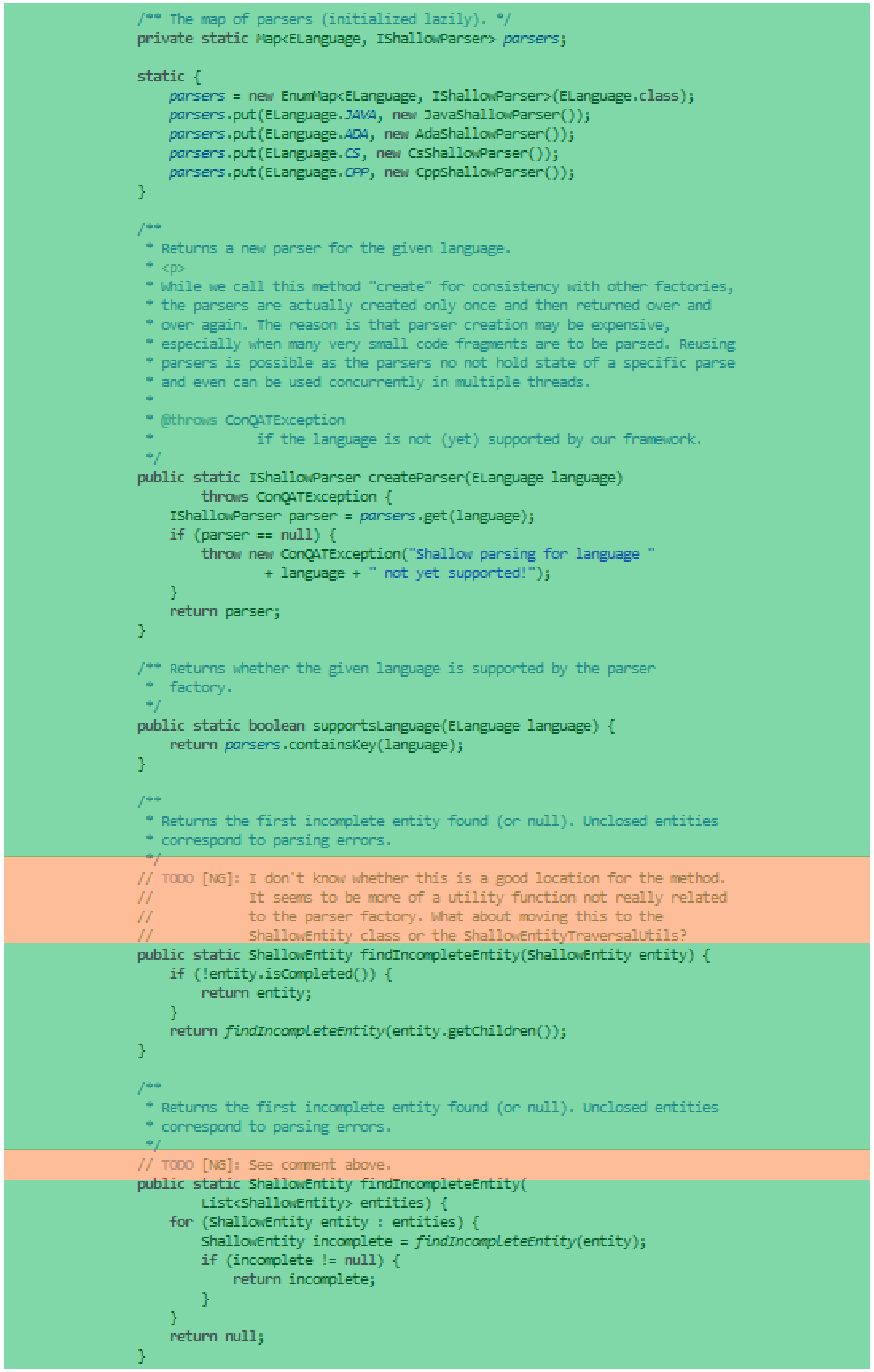

Review
Neu
RED
Request f.
Review
YELLOW
Review
GREEN
Modifizierung



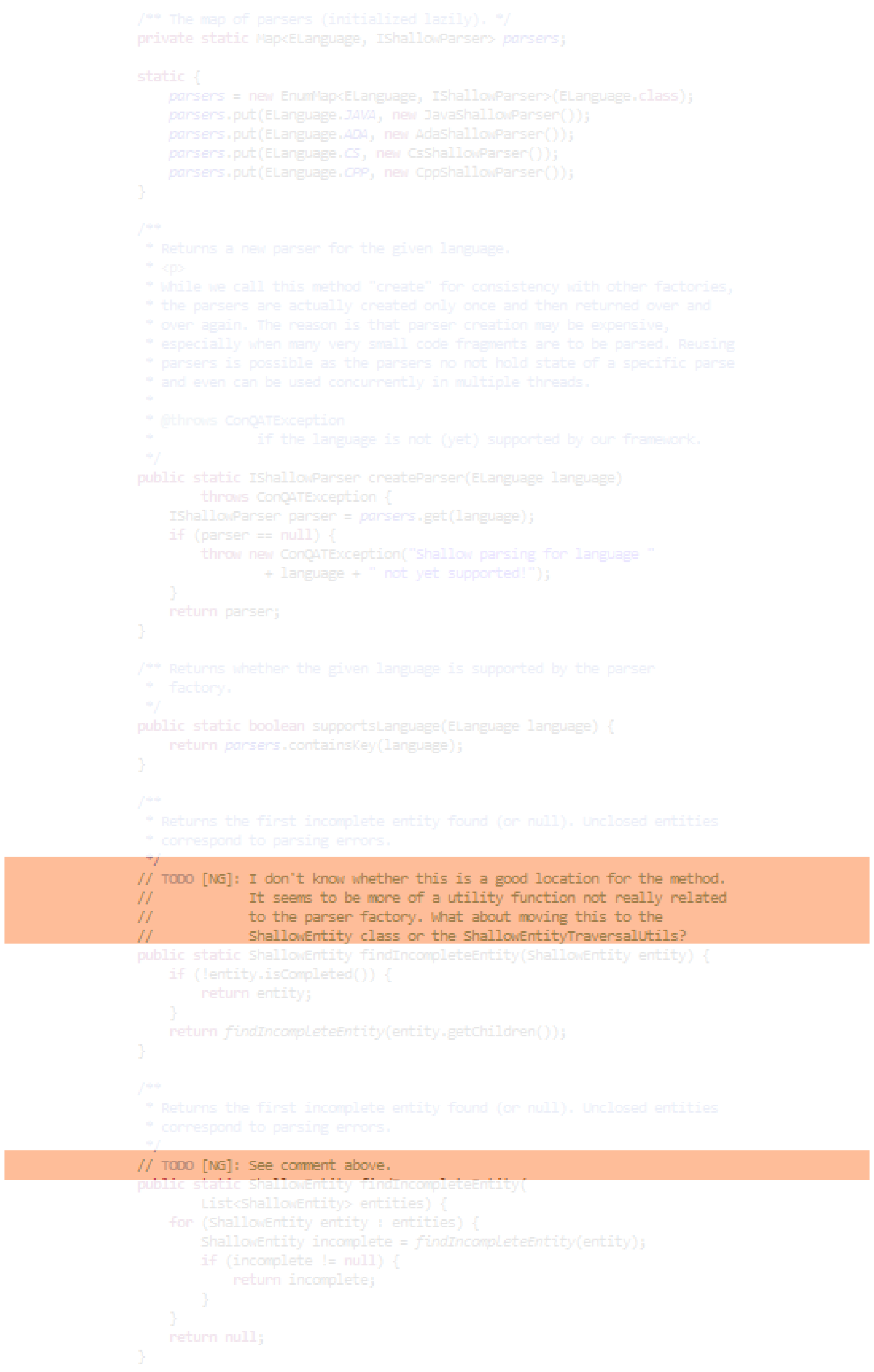
```
/**
 * Class that represents cloned code regions.
 *
 * @author $Author: hummelb $
 * @version $Rev: 36145 $
 * @ConQAT.Rating GREEN Hash: 42C76A07FE3637F87A1B3D8F2466B742
 */
```

```
f:\svnccsm\edu.tum.cs.cqedit.core\src\edu\tum\cs\cqedit\core\launching\ConQATMainTab.java
LoC        317
rating     ●
code-type  normal
```

```java
/**
 * Perform the DFS for finding the clones
 *
 * @param node
 *            the current node to search at.
 * @param currentLength
 *            the current length of the word spelled out starting from the
 *            root node.
 * @param leafPosStart
 *            the first position of the {@link #leafPositions} array which
 *            may be written.
 * @return the first position not occupied in the {@link #leafPositions}
 *         array (it is leafPosEnd).
 */
int findClones(int node, int currentLength, int leafPosStart)
        throws ConQATException {
    // is a leaf node?
    if (nodeChildFirst[node] < 0) {
        leafPositions[leafPosStart] = INFTY - currentLength;
        return leafPosStart + 1;
    }

    int leafPosEnd = leafPosStart;
    for (int e = nodeChildFirst[node]; e >= 0; e = nodeChildNext[e]) {
        int next = nodeChildNode[e];
        int len = nodeWordEnd[next] - nodeWordBegin[next];
        leafPosEnd = findClones(next, currentLength + len, leafPosEnd);
    }

    // report clones ?
    if (currentLength >= minLength
            && leafPosEnd - leafPosStart > inducedClones[node]) {
        consumer.startCloneClass(currentLength);
        for (int i = leafPosStart; i < leafPosEnd; ++i) {
            consumer.addClone(leafPositions[i], currentLength);
        }
        consumer.completeCloneClass();
    }

    return leafPosEnd;
}
```



```
/** The map of parsers (initialized lazily). */
private static Map<ELanguage, IShallowParser> parsers;

static {
    parsers = new EnumMap<ELanguage, IShallowParser>(ELanguage.class);
    parsers.put(ELanguage.JAVA, new JavaShallowParser());
    parsers.put(ELanguage.ADA, new AdaShallowParser());
    parsers.put(ELanguage.CS, new CsShallowParser());
    parsers.put(ELanguage.CPP, new CppShallowParser());
}

/**
 * Returns a new parser for the given language.
 * <p>
 * While we call this method "create" for consistency with other factories,
 * the parsers are actually created only once and then returned over and
 * over again. The reason is that parser creation may be expensive,
 * especially when many very small code fragments are to be parsed. Reusing
 * parsers is possible as the parsers no not hold state of a specific parse
 * and even can be used concurrently in multiple threads.
 *
 * @throws ConQATException
 *             if the language is not (yet) supported by our framework.
 */
public static IShallowParser createParser(ELanguage language)
        throws ConQATException {
    IShallowParser parser = parsers.get(language);
    if (parser == null) {
        throw new ConQATException("Shallow parsing for language "
                + language + " not yet supported!");
    }
    return parser;
}

/** Returns whether the given language is supported by the parser
 *  factory.
 */
public static boolean supportsLanguage(ELanguage language) {
    return parsers.containsKey(language);
}

/**
 * Returns the first incomplete entity found (or null). Unclosed entities
 * correspond to parsing errors.
 */
// TODO [NG]: I don't know whether this is a good location for the method.
//            It seems to be more of a utility function not really related
//            to the parser factory. What about moving this to the
//            ShallowEntity class or the ShallowEntityTraversalUtils?
public static ShallowEntity findIncompleteEntity(ShallowEntity entity) {
    if (!entity.isCompleted()) {
        return entity;
    }
    return findIncompleteEntity(entity.getChildren());
}

/**
 * Returns the first incomplete entity found (or null). Unclosed entities
 * correspond to parsing errors.
 */
// TODO [NG]: See comment above.
public static ShallowEntity findIncompleteEntity(
        List<ShallowEntity> entities) {
    for (ShallowEntity entity : entities) {
        ShallowEntity incomplete = findIncompleteEntity(entity);
        if (incomplete != null) {
            return incomplete;
        }
    }
    return null;
}
```



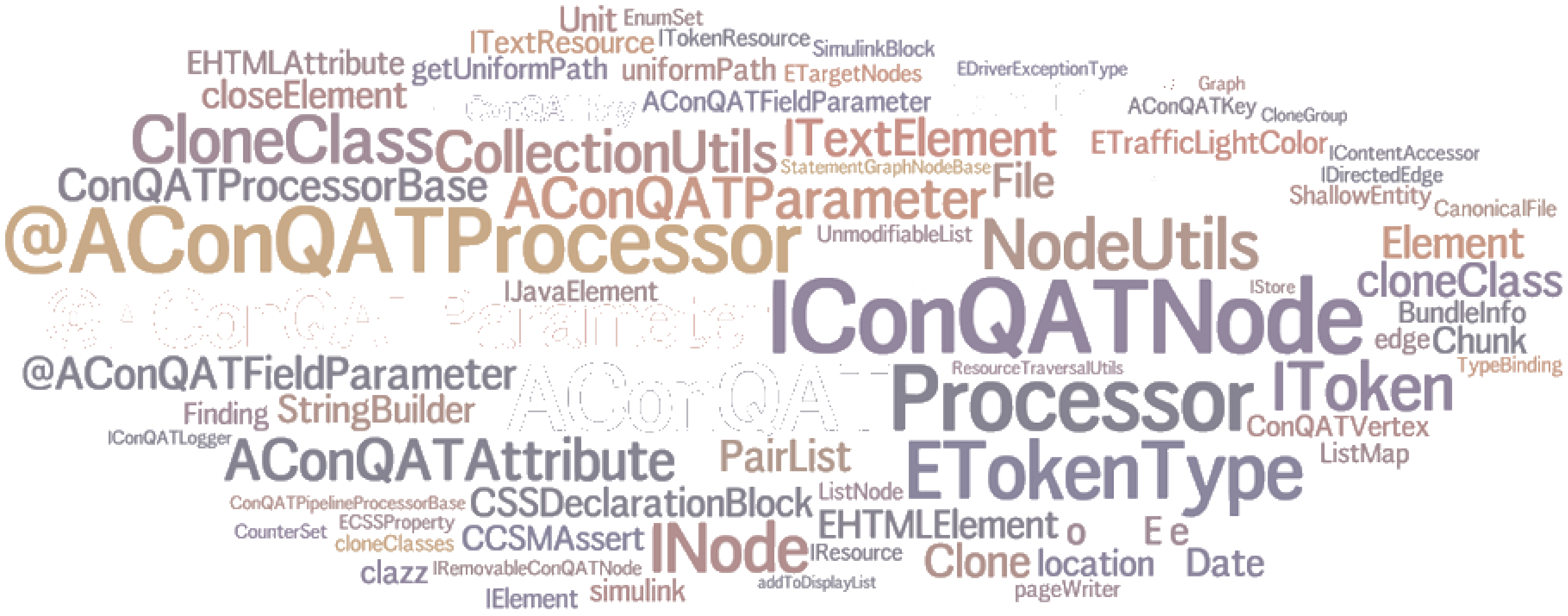
```
/** The map of parsers (initialized lazily). */
private static Map<ELanguage, IShallowParser> parsers;

static {
    parsers = new EnumMap<ELanguage, IShallowParser>(ELanguage.class);
    parsers.put(ELanguage.JAVA, new JavaShallowParser());
    parsers.put(ELanguage.ADA, new AdaShallowParser());
    parsers.put(ELanguage.CS, new CsShallowParser());
    parsers.put(ELanguage.CPP, new CppShallowParser());
}

/**
 * Returns a new parser for the given language.
 * <p>
 * While we call this method "create" for consistency with other factories,
 * the parsers are actually created only once and then returned over and
 * over again. The reason is that parser creation may be expensive,
 * especially when many very small code fragments are to be parsed. Reusing
 * parsers is possible as the parsers no not hold state of a specific parse
 * and even can be used concurrently in multiple threads.
 *
 * @throws ConQATException
 *             if the language is not (yet) supported by our framework.
 */
public static IShallowParser createParser(ELanguage language)
        throws ConQATException {
    IShallowParser parser = parsers.get(language);
    if (parser == null) {
        throw new ConQATException("Shallow parsing for language "
                + language + " not yet supported!");
    }
    return parser;
}

/** Returns whether the given language is supported by the parser
 *  factory.
 */
public static boolean supportsLanguage(ELanguage language) {
    return parsers.containsKey(language);
}

/**
 * Returns the first incomplete entity found (or null). Unclosed entities
 * correspond to parsing errors.
 */
// TODO [NG]: I don't know whether this is a good location for the method.
//            It seems to be more of a utility function not really related
//            to the parser factory. What about moving this to the
//            ShallowEntity class or the ShallowEntityTraversalUtils?
public static ShallowEntity findIncompleteEntity(ShallowEntity entity) {
    if (!entity.isCompleted()) {
        return entity;
    }
    return findIncompleteEntity(entity.getChildren());
}

/**
 * Returns the first incomplete entity found (or null). Unclosed entities
 * correspond to parsing errors.
 */
// TODO [NG]: See comment above.
public static ShallowEntity findIncompleteEntity(
        List<ShallowEntity> entities) {
    for (ShallowEntity entity : entities) {
        ShallowEntity incomplete = findIncompleteEntity(entity);
        if (incomplete != null) {
            return incomplete;
        }
    }
    return null;
}
```

Best Practice

Jeder Review-Empfänger führt selbst auch Reviews durch.

Neue Entwickler führen mindestens ein Review durch, bevor sie ein Review für ihren eigenen Code empfangen.

Unit
EnumSet
ITextResource
ITokenResource
SimulinkBlock
EHTMLAttribute
getUniformPath
uniformPath
ETargetNodes
EDriverExceptionType
closeElement
ConQATGraph
@AConQATKey
AConQATFieldParameter
Collection
AConQATKey
CloneGroup
CloneClass
CollectionUtils
ITextElement
ETrafficLightColor
IContentAccessor
StatementGraphNodeBase
ConQATProcessorBase
AConQATParameter
FileSystemUtils
IDirectedEdge
ShallowEntity
CanonicalFile
@AConQATProcessor
UnmodifiableList
ComponentNode
NodeUtils
ITokenElement
Pattern
IJavaElement
IStore
cloneClass
@AConQATParameter
IConQATNode
BundleInfo
edge
Chunk
@AConQATFieldParameter
ResourceTraversalUtils
TypeBinding
AConQATProcessor
IToken
Finding
StringBuilder
ConQATVertex
IConQATLogger
AConQATAttribute
PairList
ETokenType
ListMap
ConQATPipelineProcessorBase
ListNode
CSSDeclarationBlock
ECSSProperty
EHTMLElement
openElement
CounterSet
cloneClasses
CCSMAssert
INode
IResource
Clone
location
Date
clazz
IRemovableConQATNode
simulink
addToDisplayList
pageWriter
IElement

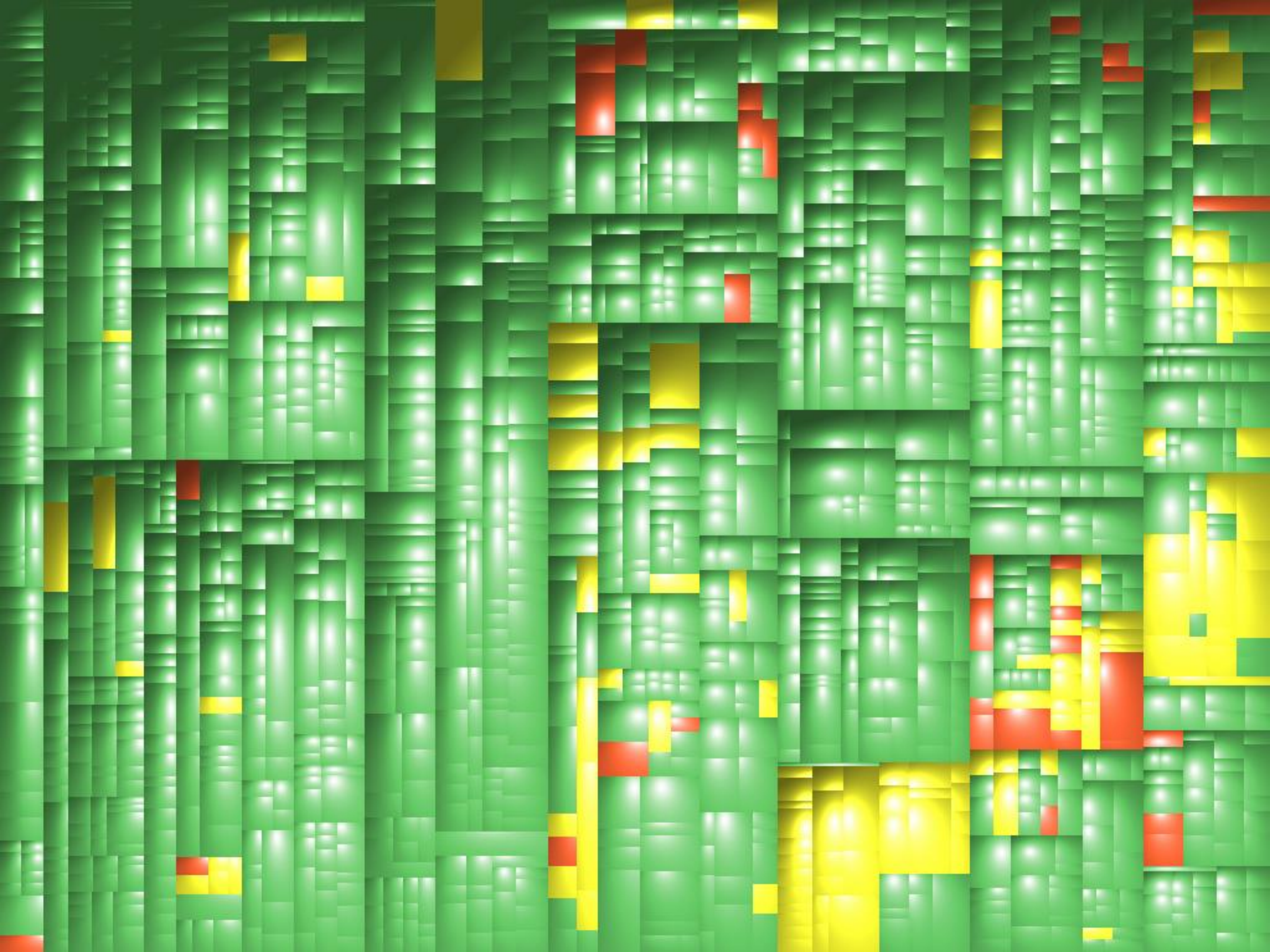

Unit
EnumSet
ITextResource
ITokenResource
SimulinkBlock
EHTMLAttribute
getUniformPath
uniformPath
ETargetNodes
EDriverExceptionType
Graph
closeElement
AConQATFieldParameter
AConQATKey
CloneGroup
CloneClass
CollectionUtils
ITextElement
ETrafficLightColor
IContentAccessor
StatementGraphNodeBase
IDirectedEdge
ConQATProcessorBase
AConQATParameter
File
ShallowEntity
CanonicalFile
@AConQATProcessor
UnmodifiableList
NodeUtils
Element
IJavaElement
IStore
cloneClass
@AConQATParameter
IConQATNode
BundleInfo
edge
Chunk
@AConQATFieldParameter
AConQAT
ResourceTraversalUtils
TypeBinding
Processor
IToken
Finding
StringBuilder
ConQATVertex
IConQATLogger
AConQATAttribute
PairList
ListMap
ETokenType
ConQATPipelineProcessorBase
ListNode
CSSDeclarationBlock
ECSSProperty
EHTMLElement
CounterSet
o
E
e
cloneClasses
CCSMAssert
INode
IResource
Clone
location
Date
clazz
IRemovableConQATNode
addToDisplayList
pageWriter
simulink
IElement

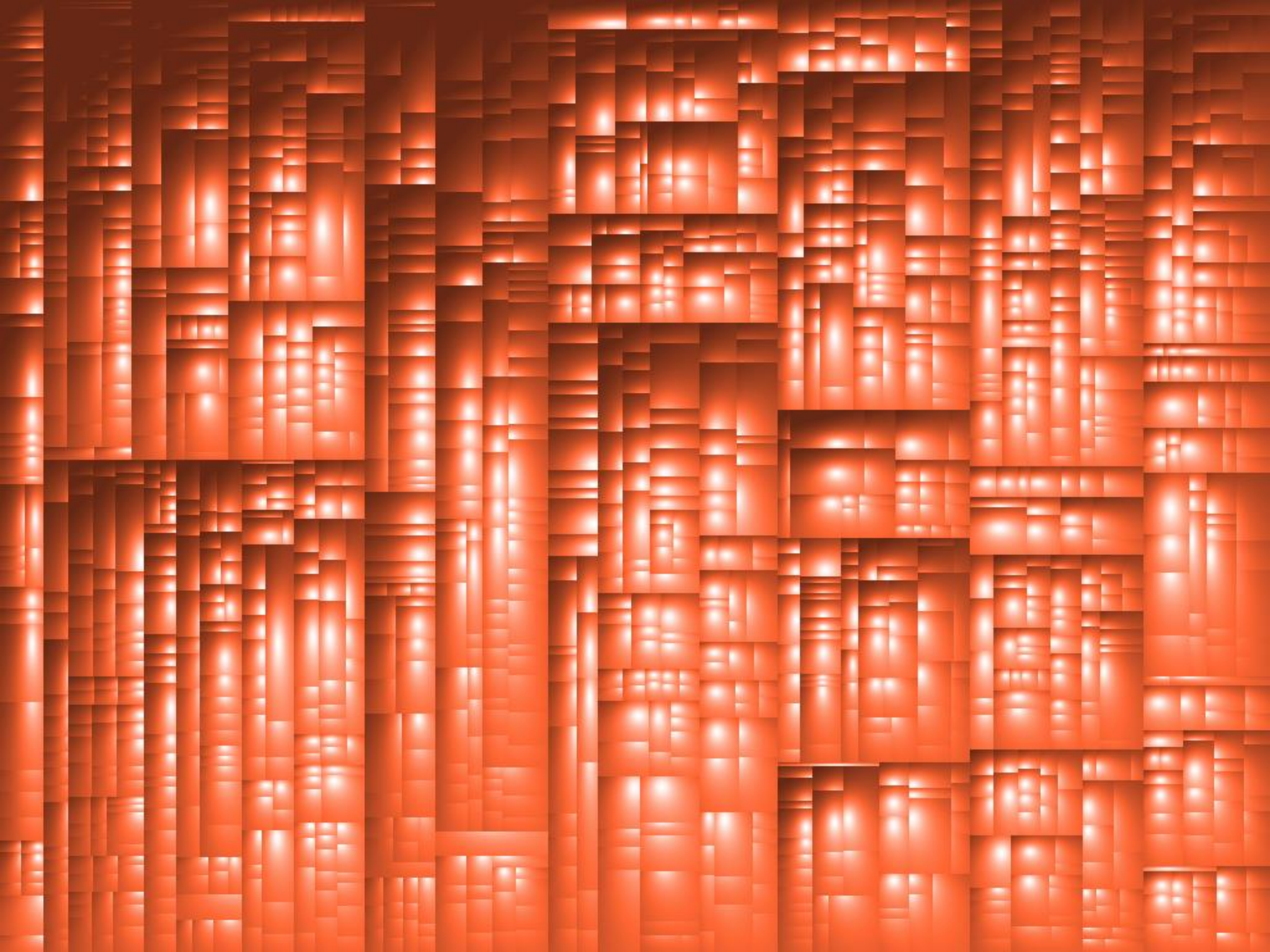
Best Practice

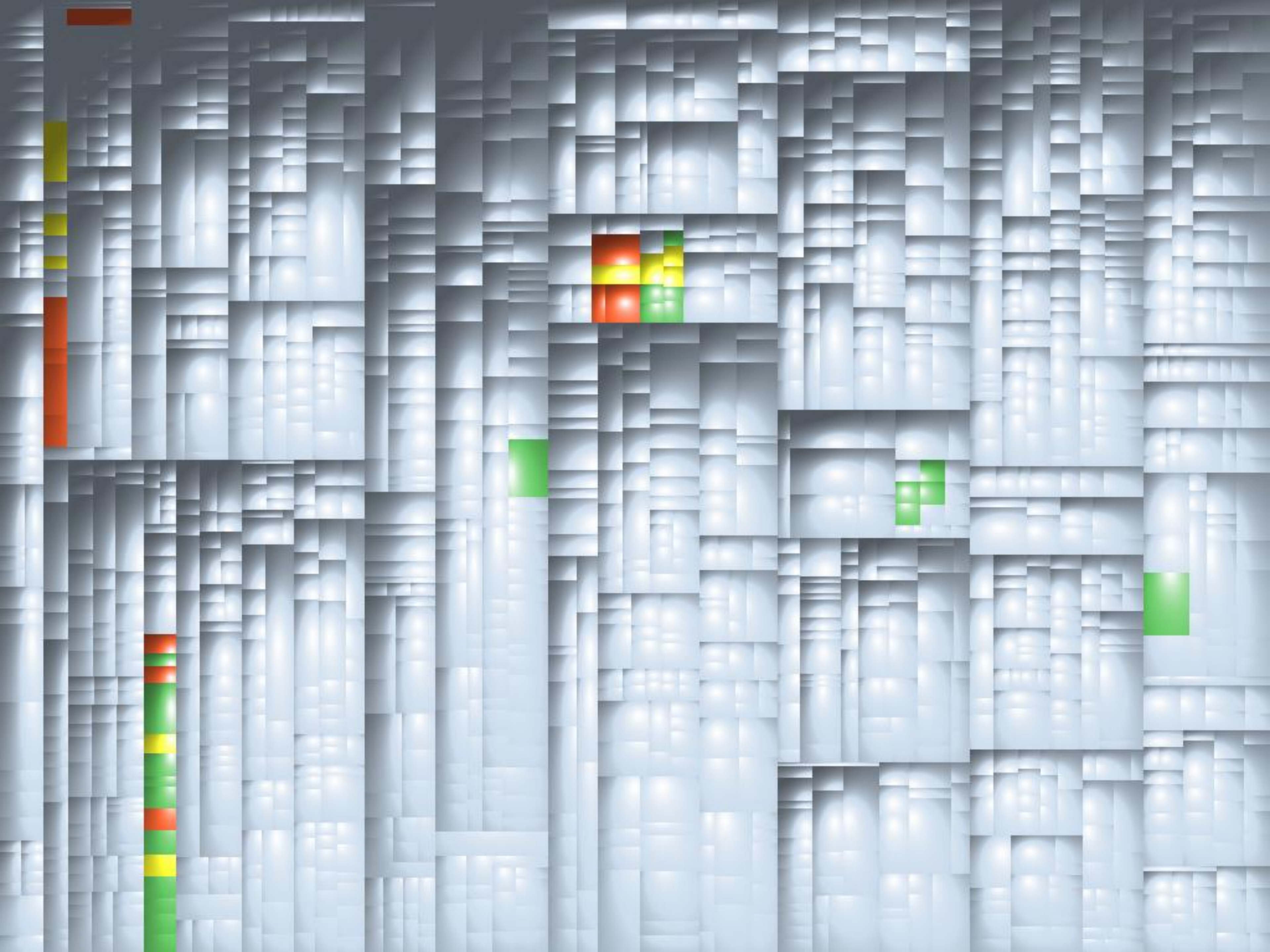
Wenn der Reviewer den Code nicht versteht, dann hat der Reviewer recht. Nicht der Entwickler.

Best Practice

Es dürfen nur Reviews für Code angefordert werden, der keine Probleme enthält, die automatisierte statische Analysen aufdecken können.







Best Practice

Alten und neuen Code unterscheiden.

Weitere Best-Practices

- Reviewer setzt Verbesserungen i.d.R. nicht selbst um
- Review-Blocker 1x pro Woche (Freitag Vormittag)
- Incubator Bereiche
- Diskussionen persönlich führen (nicht im Code)
- Guidelines und Checks kontinuierlich pflegen
- 2-Level Review bei Einarbeitung neuer Mitarbeiter
- …

Fazit

Effektive statische Analysen sind die Voraussetzung für effiziente Peer-Reviews.

Peer-Reviews sind der Schlüssel zu wartbarer Software.

Erfolgreiche Peer-Reviews erfordern allerdings hohes Commitment, inklusive Ressourcen, auf allen Ebenen.

Kontakt

Ich freue mich auf Diskussionen ☺

Dr. Elmar Jürgens · juergens@cqse.eu · +49 179 675 3863

@ElmarJuergens
@teamscale
www.cqse.eu/en/blog

CQSE GmbH, Lichtenbergstraße 8,
85748 Garching bei München

CQSE
Continuous Quality in Software Engineering