Technische Schulden

einschätzen und wirksam kommunizieren

CQSE

Dr. Nils Göde
Dr. Elmar Jürgens

Praxis

Software-**Audits**

Kontinuierliche **Qualitäts-** und **Testkontrolle**

Teamscale

CQSE GmbH

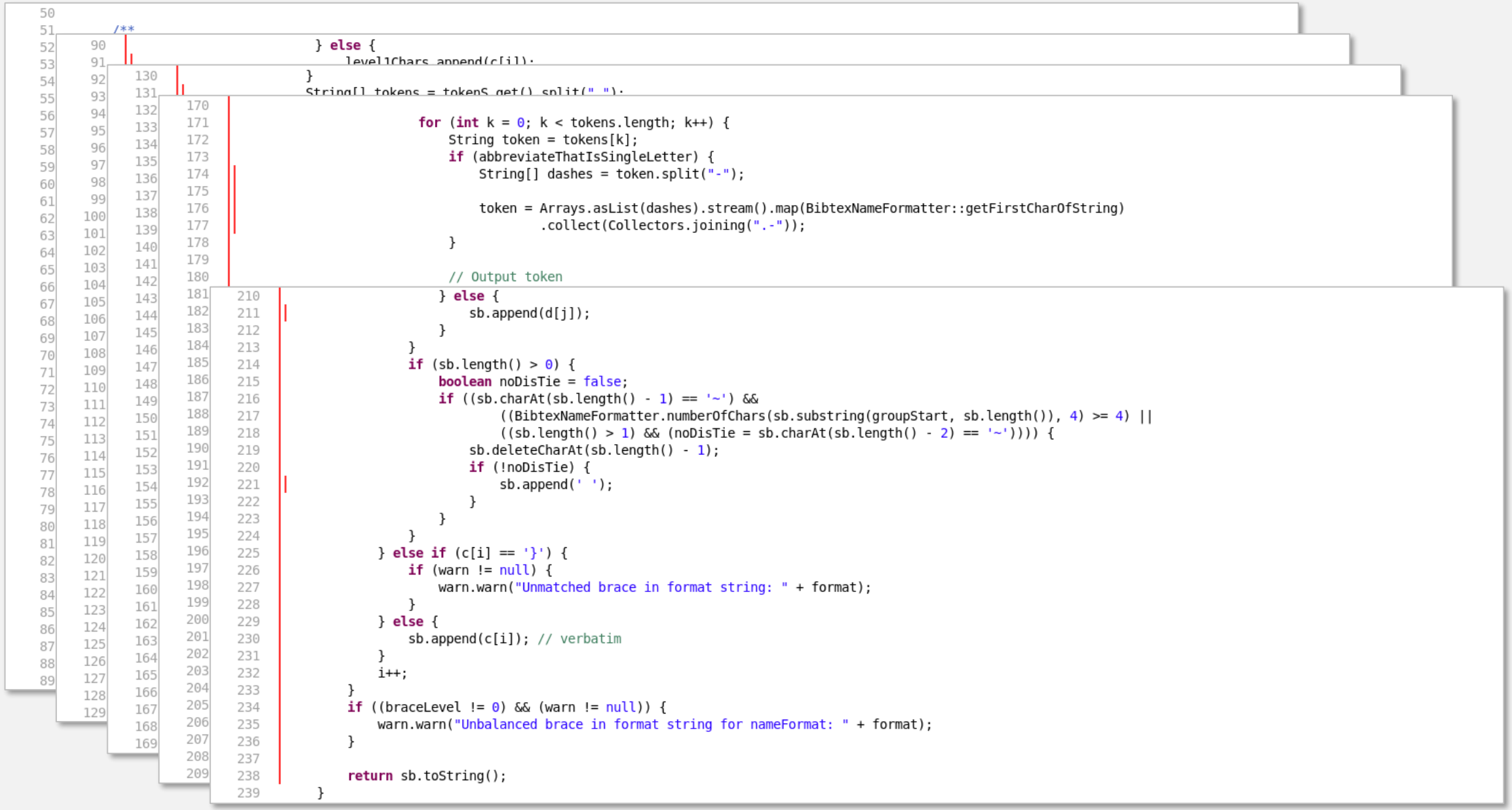
Forschung

18+ **Promotionen** in Software Engineering

Eigene **Forschung**

Enger Kontakt zu **Universitäten**

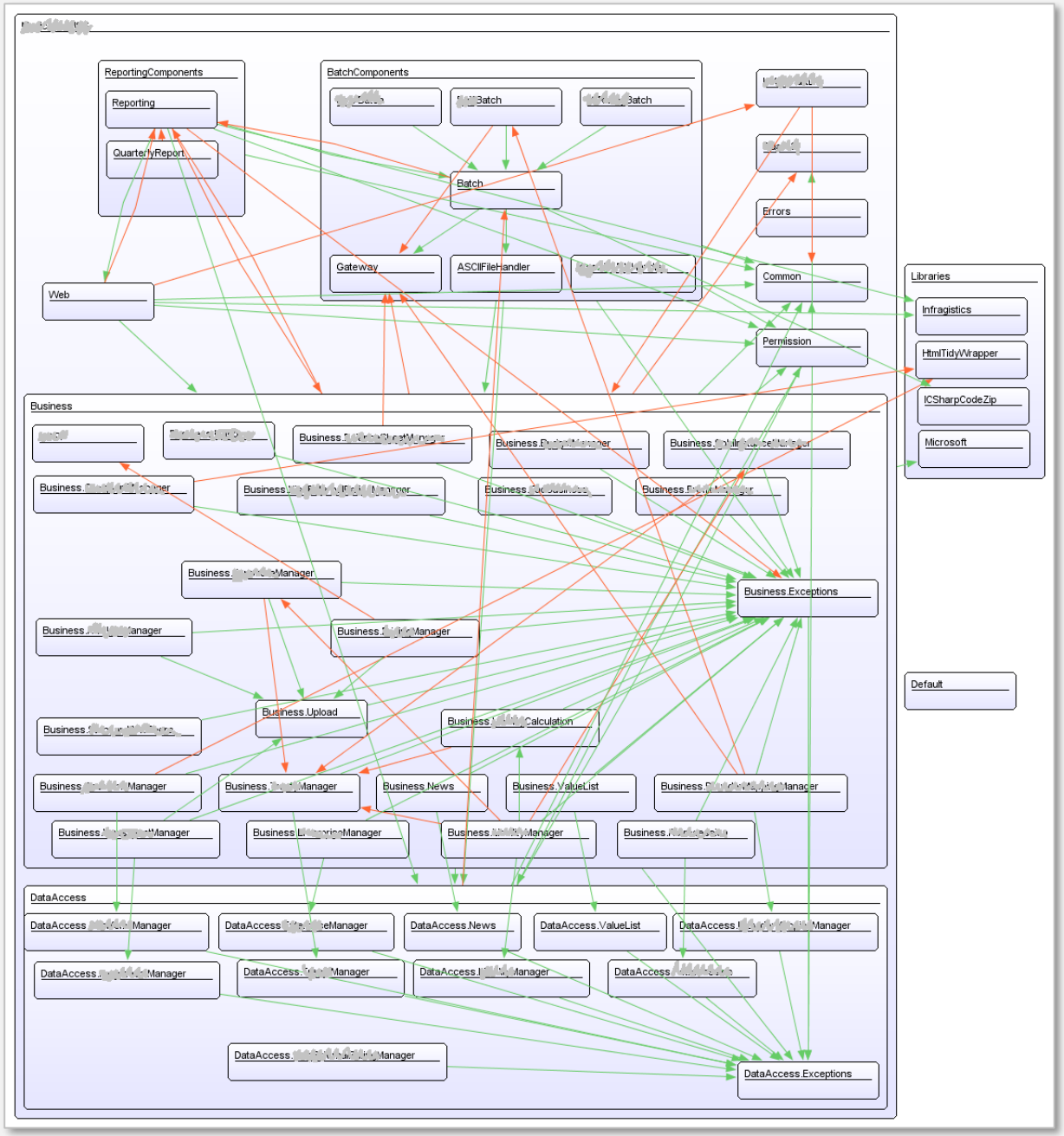
Technische Schulden



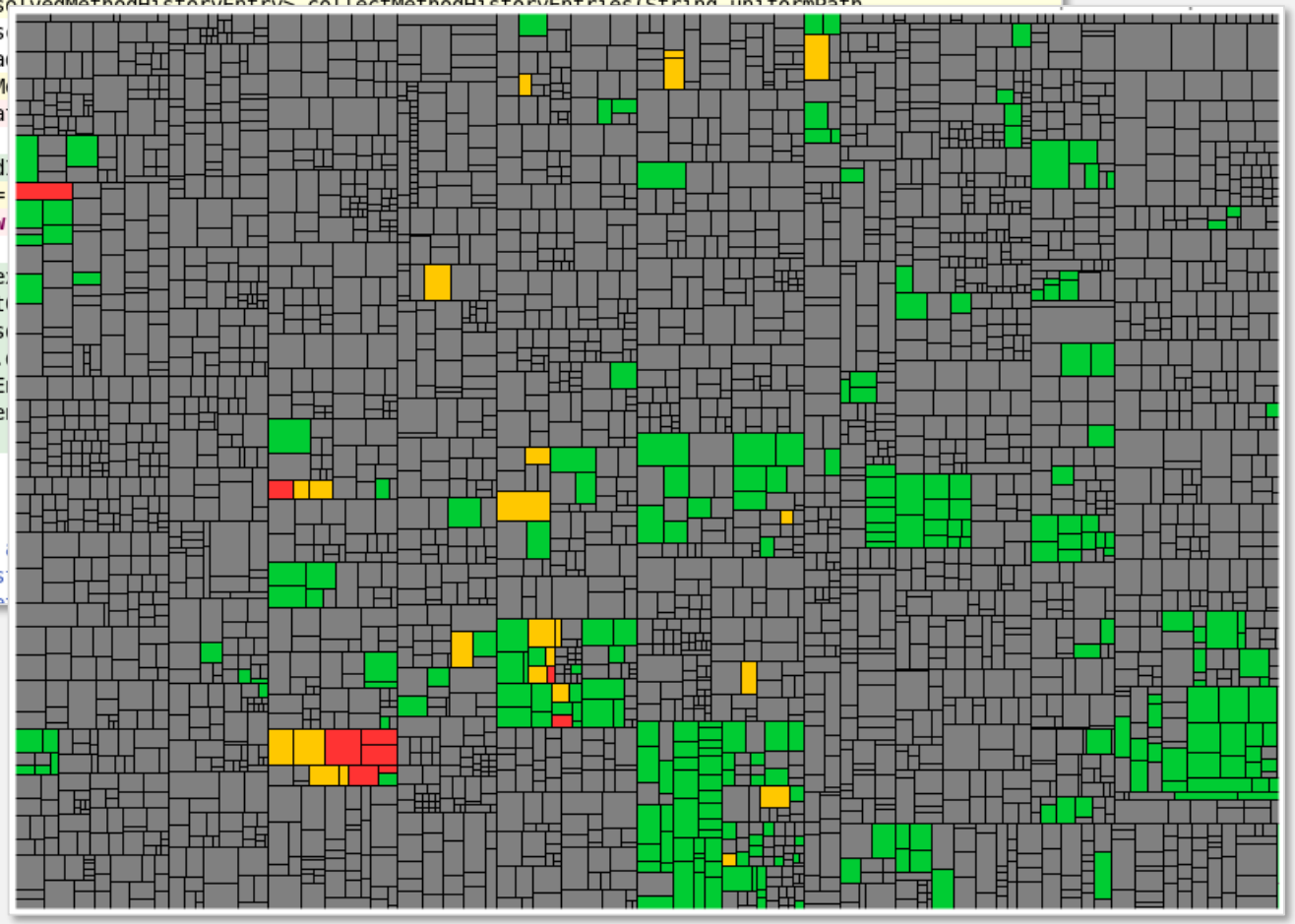
```
for (int j = 0; j < d.length; j++) {

    if (Character.isLetter(d[j]) && (bLevel == 1)) {
        groupStart = sb.length();
        if (!abbreviateThatIsSingleLetter) {
            j++;
        }
        if (((j + 1) < d.length) && (d[j + 1] == '{')) {
            StringBuilder interTokenSb = new StringBuilder();
            j = BibtexNameFormatter.consumeToMatchingBrace(interTokenSb, d, j + 1);
            interToken = interTokenSb.substring(1, interTokenSb.length() - 1);
        }

        for (int k = 0; k < tokens.length; k++) {
            String token = tokens[k];
            if (abbreviateThatIsSingleLetter) {
                String[] dashes = token.split("-");

                token = Arrays.asList(dashes).stream().map(BibtexNameFormatter::getFirstCharOfString)
                        .collect(Collectors.joining(".-"));
            }

            // Output token
            sb.append(token);

            if (k < (tokens.length - 1)) {
                // Output Intertoken String
                if (interToken == null) {
                    if (abbreviateThatIsSingleLetter) {
                        sb.append('.');
                    }
                    // No clue what this means (What the hell are tokens anyway???
                    // if (lex_class[name_sep_char[cur_token]] = sep_char) then
                    //    append_ex_buf_char_and_check (name_sep_char[cur_token])
                    if ((k == (tokens.length - 2)) || (BibtexNameFormatter.numberOfChars(sb.substring(groupStart, sb.length()), 3) < 3)) {
                        sb.append('~');
                    } else {
                        sb.append(' ');
                    }
                } else {
                    sb.append(interToken);
                }
            }
        }
    } else if (d[j] == '}') {
        bLevel--;
        if (bLevel > 0) {
            sb.append('}');
        }
    } else if (d[j] == '{') {
        bLevel++;
        sb.append('{');
    } else {
        sb.append(d[j]);
    }
}
if (sb.length() > 0) {
```

```
                        i++;
                        c = finalResult.charAt(i);
                        String combody;
                        if (c == '{') {
                            String part = StringUtil.getPart(finalResult, i, false);
                            i += part.length();
                            combody = part;
                        } else {
                            combody = finalResult.substring(i, i + 1);
                        }
                        String result = OOPreFormatter.CHARS.get(command + combody);

                        if (result != null) {
                            sb.append(result);
                        }

                        incommand = false;
                        escaped = false;
                    } else {
                        //    Are we already at the end of the string?
                        if ((i + 1) == finalResult.length()) {
                            String command = currentCommand.toString();
                            String result = OOPreFormatter.CHARS.get(command);
                            /* If found, then use translated version. If not,
                             * then keep
                             * the text of the parameter intact.
                             */
                            if (result == null) {
                                sb.append(command);
                            } else {
                                sb.append(result);
                            }

                        }
                    }
                }
            } else {
```

```
                        i++;
                        c = field.charAt(i);
                        String commandBody;
                        if (c == '{') {
                            String part = StringUtil.getPart(field, i, false);
                            i += part.length();
                            commandBody = part;
                        } else {
                            commandBody = field.substring(i, i + 1);
                        }
                        String result = HTML_CHARS.get(command + commandBody);

                        if (result == null) {
                            sb.append(commandBody);
                        } else {
                            sb.append(result);
                        }

                        incommand = false;
                        escaped = false;
                    } else {
                        //    Are we already at the end of the string?
                        if ((i + 1) == field.length()) {
                            String command = currentCommand.toString();
                            String result = HTML_CHARS.get(command);
                            /* If found, then use translated version. If not,
                             * then keep
                             * the text of the parameter intact.
                             */
                            if (result == null) {
                                sb.append(command);
                            } else {
                                sb.append(result);
                            }

                        }
                    }
                }
            } else {
```


ReportingComponents
Reporting
QuarterlyReport
BatchComponents
Batch
Gateway
ASCIIFileHandler
Web
Errors
Common
Permission
Libraries
Infragistics
HtmlTidyWrapper
ICSharpCodeZip
Microsoft
Business
Business.Exceptions
Business.Upload
Business.News
Business.ValueList
Default
DataAccess
DataAccess.News
DataAccess.ValueList
DataAccess.Exceptions

```
                                                    entry -> !entry.getRepositoryIdentifier().equals(EXTERNAL_ANALYSIS_REPOSI
146                         return serializeObjectToHttp(result, query);
147                     }
148                     return serializeObjectToHttp(entries, query);
149             } catch (ConQATException e) {
150                     throw new InternalServiceException(e.getMessage(), e);
151             }
152     }
153
154     /**
155      * Adds method history entries for the given parameters. Will throw an exception
156      * if no method info could be found for the given data.
157      */
158     private SortedSet<UserResolvedMethodHistoryEntry> collectMethodHistoryEntries(String uniformPath
159                     CommitDes
160                     throws Ba
161             if (ArchitectureM
162                     uniformPa
163             }
164             MethodInfo method
165             if (methodInfo ==
166                     throw new
167             }
168             repositoryLogInde
169             branchesThatMight
170             SortedSet<UserRes
171                             .
172             addMethodHistoryE
173                             e
174             return entries;
175     }
176
177     /**
178      * Uses the given method
179      * until the method has s
180      * modification of the me
```



Findings

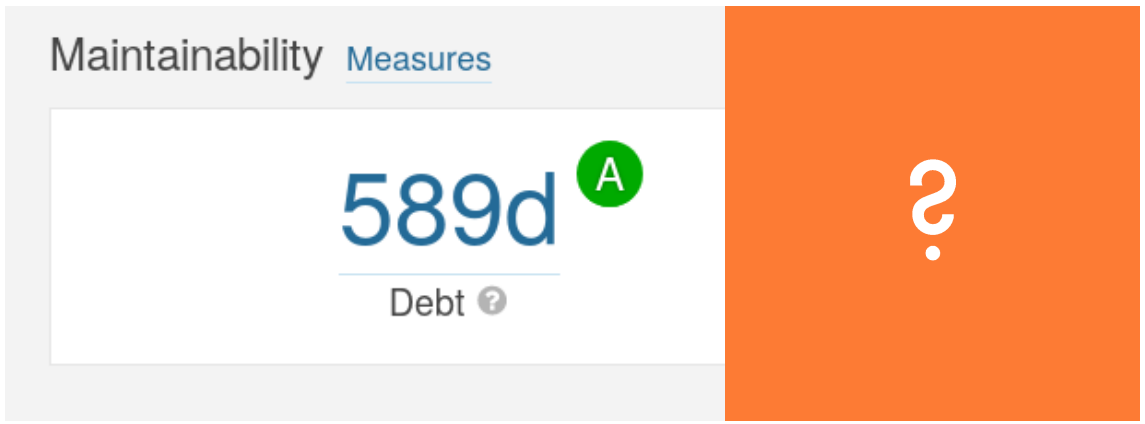
All	**6793**
Architecture	**1**
Architecture Conformance	1
Code Anomalies	**1344**
Bad practice	971
Correctness	2
Exception Handling	62
General checks (built-in)	120
Null pointer dereference	13
Performance	36
Unused code	93
Unused variable or parameter	47
Code Duplication	**988**
Cloning	101
Redundant Literals	887
Documentation	**3378**
Comment completeness	3236
Task tags	142
Formatting	**6**
Code formatting	6
Naming	**110**
Java naming conventions	110
Structure	**966**
File Size	38
Method Length	278
Nesting Depth	650

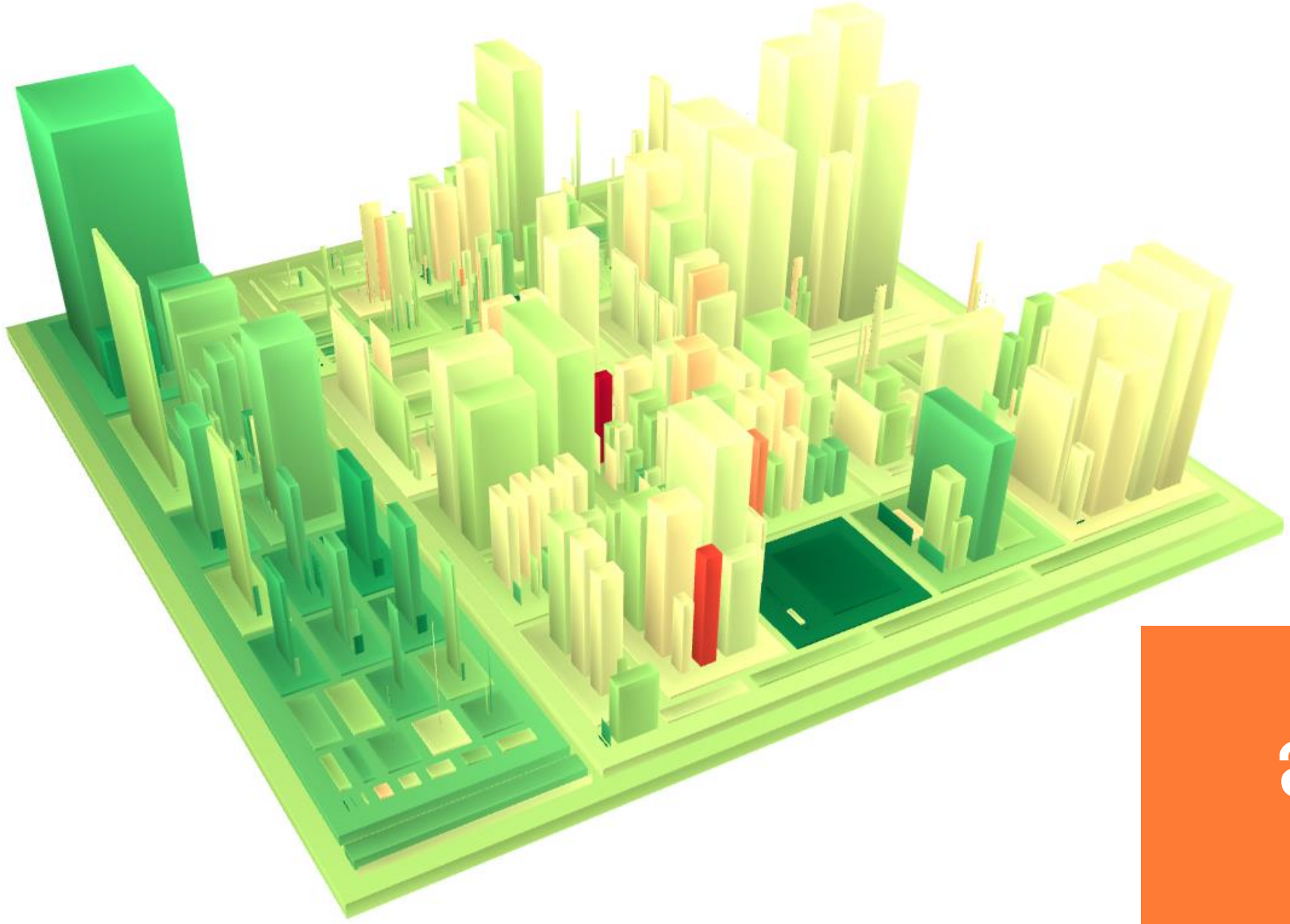
Und jetzt?

$$171 - 5.2 \cdot \ln(avgHV) - 0.23 \cdot \text{avgCC}(g') - 16.2 \cdot \ln(avgLOC) + 50 \cdot \sin(\text{sqrt}(2.4 \cdot perCM))$$

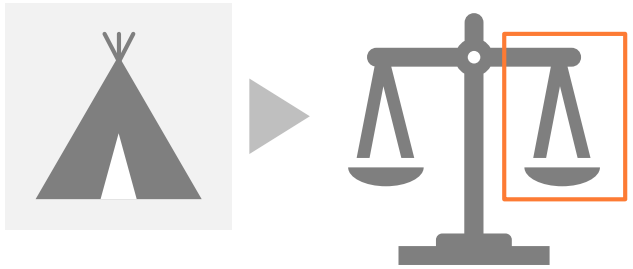
HV:	Halstead Volume	CC:	Cyclomatic Complexity
LOC:	lines of code	perCM:	% Comment Lines

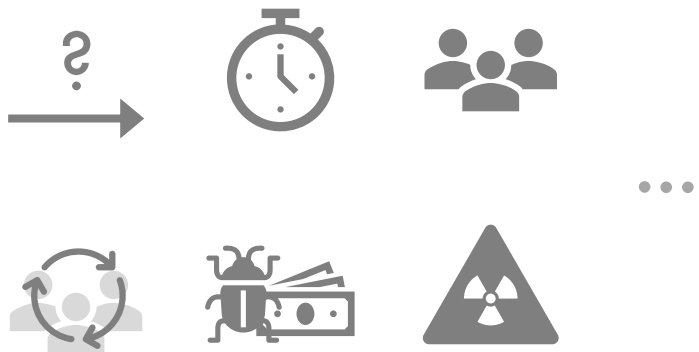
?


Maintainability
Measures
589d
A
Debt
?



?





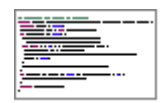
Findings

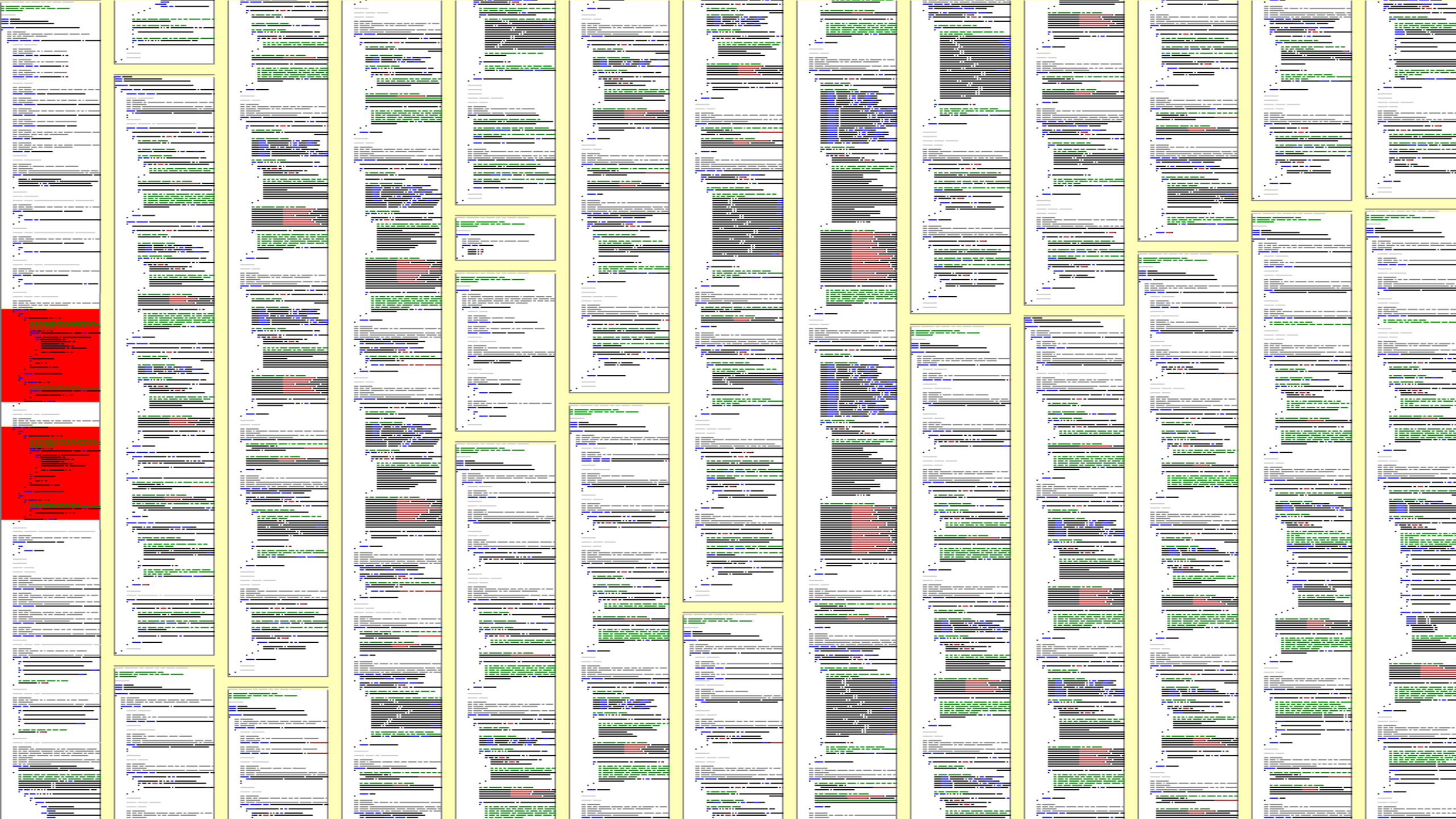
All	**6793**
Architecture	**1**
Architecture Conformance	1
Code Anomalies	**1344**
Bad practice	971
Correctness	2
Exception Handling	62
General checks (built-in)	120
Null pointer dereference	13
Performance	36
Unused code	93
Unused variable or parameter	47
Code Duplication	**988**
Cloning	101
Redundant Literals	887
Documentation	**3378**
Comment completeness	3236
Task tags	142
Formatting	**6**
Code formatting	6
Naming	**110**
Java naming conventions	110
Structure	**966**
File Size	38
Method Length	278
Nesting Depth	650

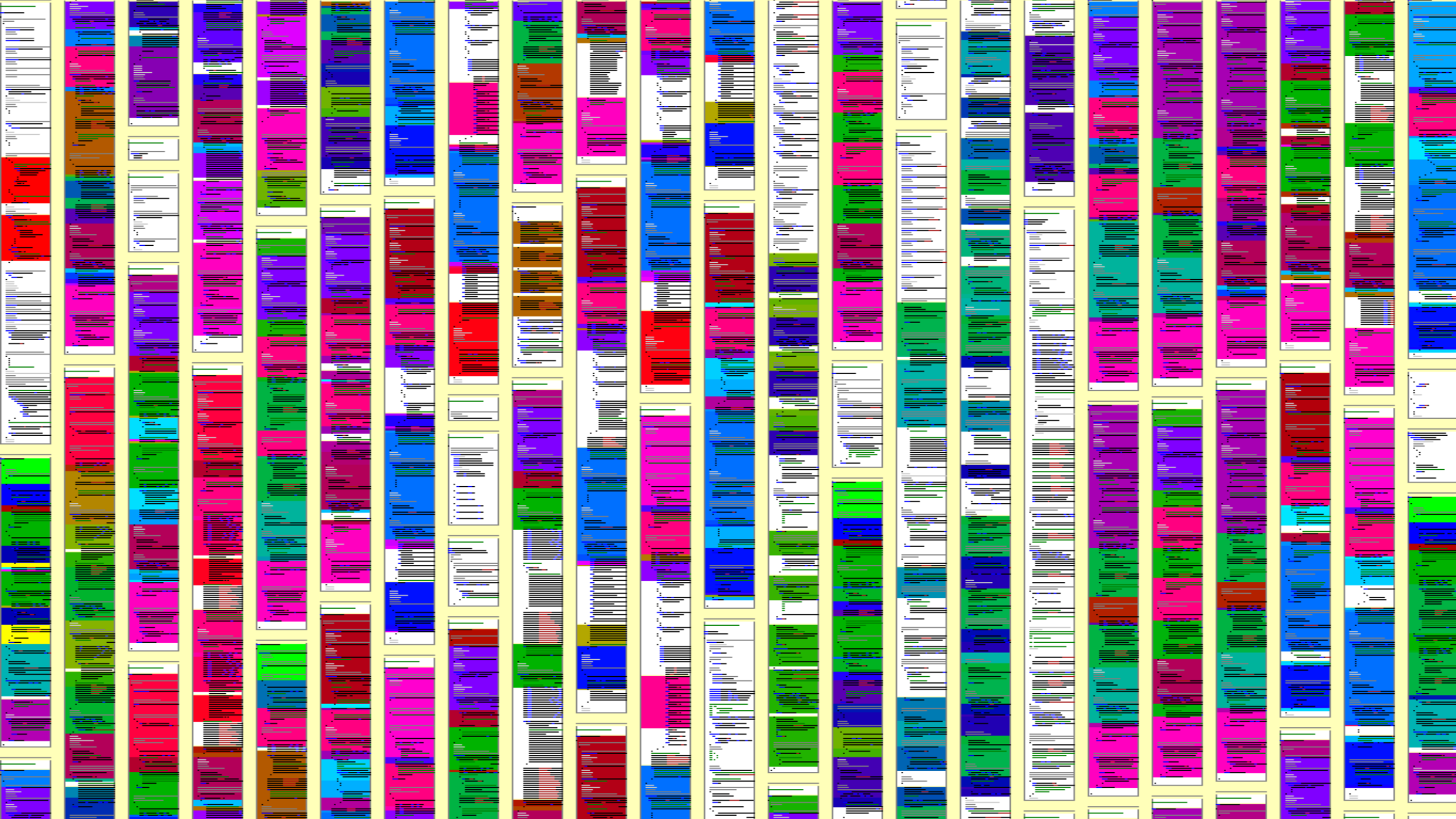
```java
// Utilities for arrays of elements
public String showElements(ModelElement[] elements, String nomsg) {
   boolean found = false;
   StringBuffer res = new StringBuffer();
   if (elements != null) {
      Index.getInstance().setCurrentRenderer(
            FlatReferenceRenderer.getInstance());
      for (int i = 0; i < elements.length; i++) {
         ModelElement el = elements[i];
         res.append(showElementLink(el)).append(HTML.LINE_BREAK);
         found = true;
      }
      Index.getInstance().resetCurrentRenderer();
   }
   if (!found && nomsg != null && nomsg.length() > 0) {
      res.append(HTML.italics(nomsg));
   }
   return res.toString();
}
```

```java
// Utilities for arrays of elements
public String showElements(ModelElement[] elements, String nomsg) {
   boolean found = false;
   StringBuffer res = new StringBuffer();
   if (elements != null) {
      Index.getInstance().setCurrentRenderer(
            FlatReferenceRenderer.getInstance());
      for (int i = 0; i < elements.length; i++) {
         ModelElement el = elements[i];
         res.append(showElementLink(el)).append(HTML.LINE_BREAK);
         found = true;
      }
      Index.getInstance().resetCurrentRenderer();
   }
   if (!found && nomsg.length() > 0) {
      res.append(HTML.italics(nomsg));
   }
   return res.toString();
}
```

```java
// Utilities for arrays of elements
public String showElements(ModelElement[] elements, String nomsg) {
    boolean found = false;
    StringBuffer res = new StringBuffer();
    if (elements != null) {
        Index.getInstance().setCurrentRenderer(
                FlatReferenceRenderer.getInstance());
        for (int i = 0; i < elements.length; i++) {
            ModelElement el = elements[i];
            res.append(showElementLink(el)).append(HTML.LINE_BREAK);
            found = true;
        }
        Index.getInstance().resetCurrentRenderer();
    }
    if (!found && nomsg != null && nomsg.length() > 0) {
        res.append(HTML.italics(nomsg));
    }
    return res.toString();
}
```







```java
// Utilities for arrays of elements
public String showElements(ModelElement[] elements, String nomsg) {
   boolean found = false;
   StringBuffer res = new StringBuffer();
   if (elements != null) {
      Index.getInstance().setCurrentRenderer(
            FlatReferenceRenderer.getInstance());
      for (int i = 0; i < elements.length; i++) {
         ModelElement el = elements[i];
         res.append(showElementLink(el)).append(HTML.LINE_BREAK);
         found = true;
      }
      Index.getInstance().resetCurrentRenderer();
   }
   if (!found && nomsg != null && nomsg.length() > 0) {
      res.append(HTML.italics(nomsg));
   }
   return res.toString();
}
```

```java
// Utilities for arrays of elements
public String showElements(ModelElement[] elements, String nomsg) {
   boolean found = false;
   StringBuffer res = new StringBuffer();
   if (elements != null) {
      Index.getInstance().setCurrentRenderer(
            FlatReferenceRenderer.getInstance());
      for (int i = 0; i < elements.length; i++) {
         ModelElement el = elements[i];
         res.append(showElementLink(el)).append(HTML.LINE_BREAK);
         found = true;
      }
      Index.getInstance().resetCurrentRenderer();
   }
   if (!found && nomsg.length() > 0) {
      res.append(HTML.italics(nomsg));
   }
   return res.toString();
}
```

Studie

Munich Re LV 1871

- Über 100 Fehler in produktiver Software

17	44	46
Kritisch	Nutzersichtbar	Nicht nutzersichtbar

- 52% aller ungewollten Unterschiede fehlerhaft

Juergens, Deissenboeck et al. *Do Code Clones Matter?* ICSE 2009

$$\text{Anzahl } \frac{Fehler}{Jahr} \times \text{Fehlerfolgekosten } \frac{PT}{Fehler}$$

$$\text{Anzahl } \frac{Fehler}{Jahr} \times \text{Fehlerfolgekosten } \frac{PT}{Fehler}$$

#Fehler durch inkonsistente Klone @ Munich Re

Daten aus Studie

- 3 Systeme von Munich Re analysiert
- 79 Fehler gefunden (Impact auf Funktionalität, nicht nur Wartbarkeit o.ä.)
- System waren produktiv, einzelne Fehler schon durch Anwender als Tickets reportet
- 1 Produktionsfehler durch inkonsistente Klone / 17k SLOC

Bedeutung heute

- Betrachtetes Portfolio der Munich Re umfasst ca. 8,25 Millionen SLOC
- Konservative Annahme: Clone Management spart 1 Produktionsfehler pro 50k SLOC pro Jahr
- 8,25 Millionen SLOC / 50k = 165

$$\text{Anzahl } \frac{Fehler}{Jahr} \times \text{Fehlerfolgekosten } \frac{PT}{Fehler}$$

$$165\ \frac{Fehler}{Jahr} \times \text{Fehlerfolgekosten}\ \frac{PT}{Fehler}$$

$$165\ \frac{Fehler}{Jahr} \times \textbf{Fehlerfolgekosten}\ \frac{PT}{Fehler}$$

ØFehlerfolgekosten von Fehlern in Produktion

Mögliche Auswirkungen fehlerhafter Software

- Nutzer bekommen falsche Ergebnisse
- Anwendung stürzt ab
- Daten gehen verloren
- Frustration bei Nutzern (Kunden und Mitarbeiter)

? PT

Aufwand für Reparatur

- Nutzer schreibt Ticket für Fehler
- Debugging (Nachstellen, Diagnose, …)
- Fixing
- Test
- Ggf. Deployment

? PT

ØFehlerfolgekosten von Fehlern in Produktion

Mögliche Auswirkungen fehlerhafter Software

- Nutzer bekommen falsche Ergebnisse
- Anwendung stürzt ab
- Daten gehen verloren
- Frustration bei Nutzern (Kunden und Mitarbeiter)

0 PT: bewusste Unterschätzung

Aufwand für Reparatur

- Nutzer schreibt Ticket für Fehler
- Debugging (Nachstellen, Diagnose, …)
- Fixing
- Test
- Ggf. Deployment

3 PT

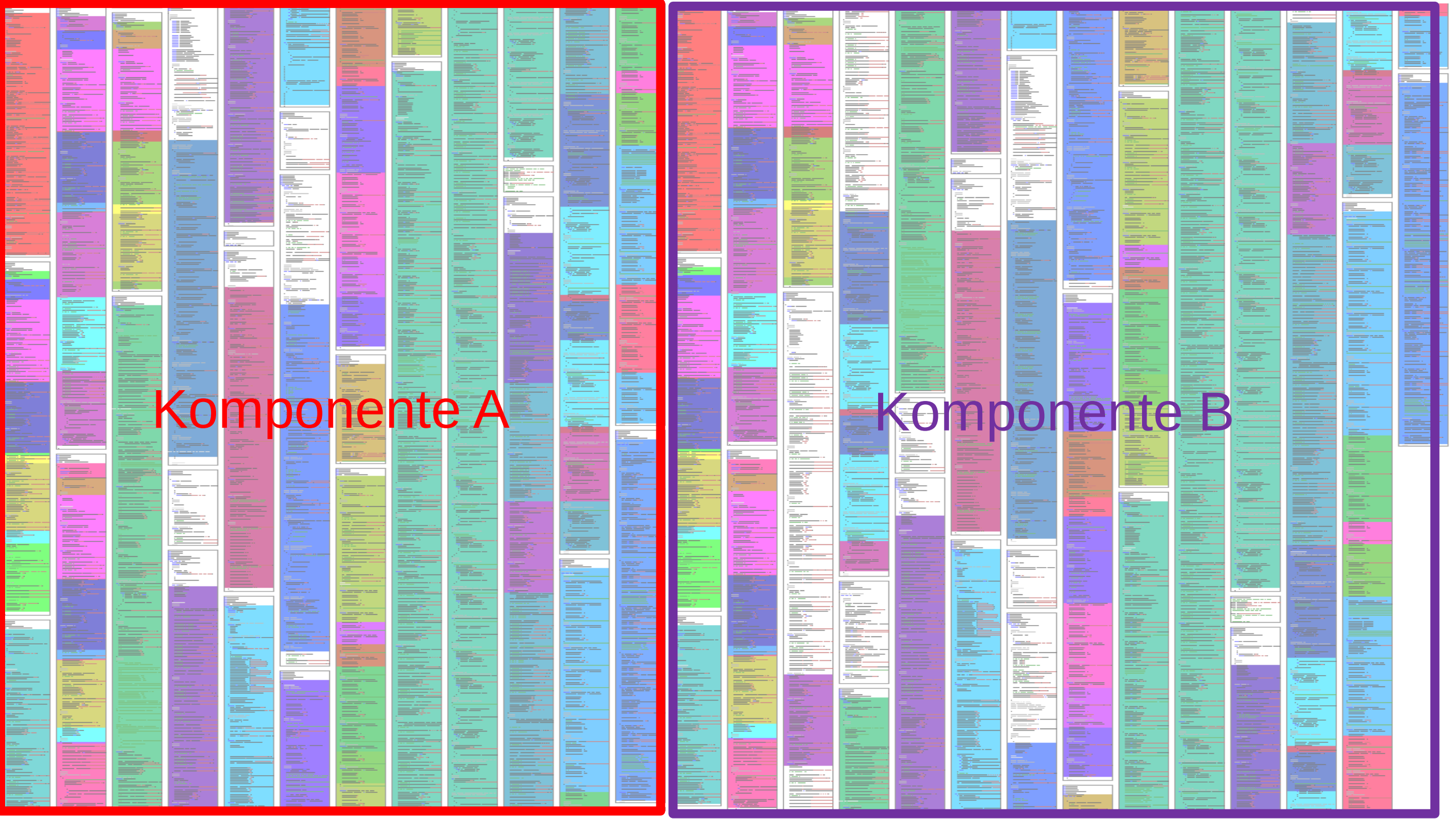
$$165\ \frac{Fehler}{Jahr} \times \textbf{Fehlerfolgekosten}\ \frac{PT}{Fehler}$$

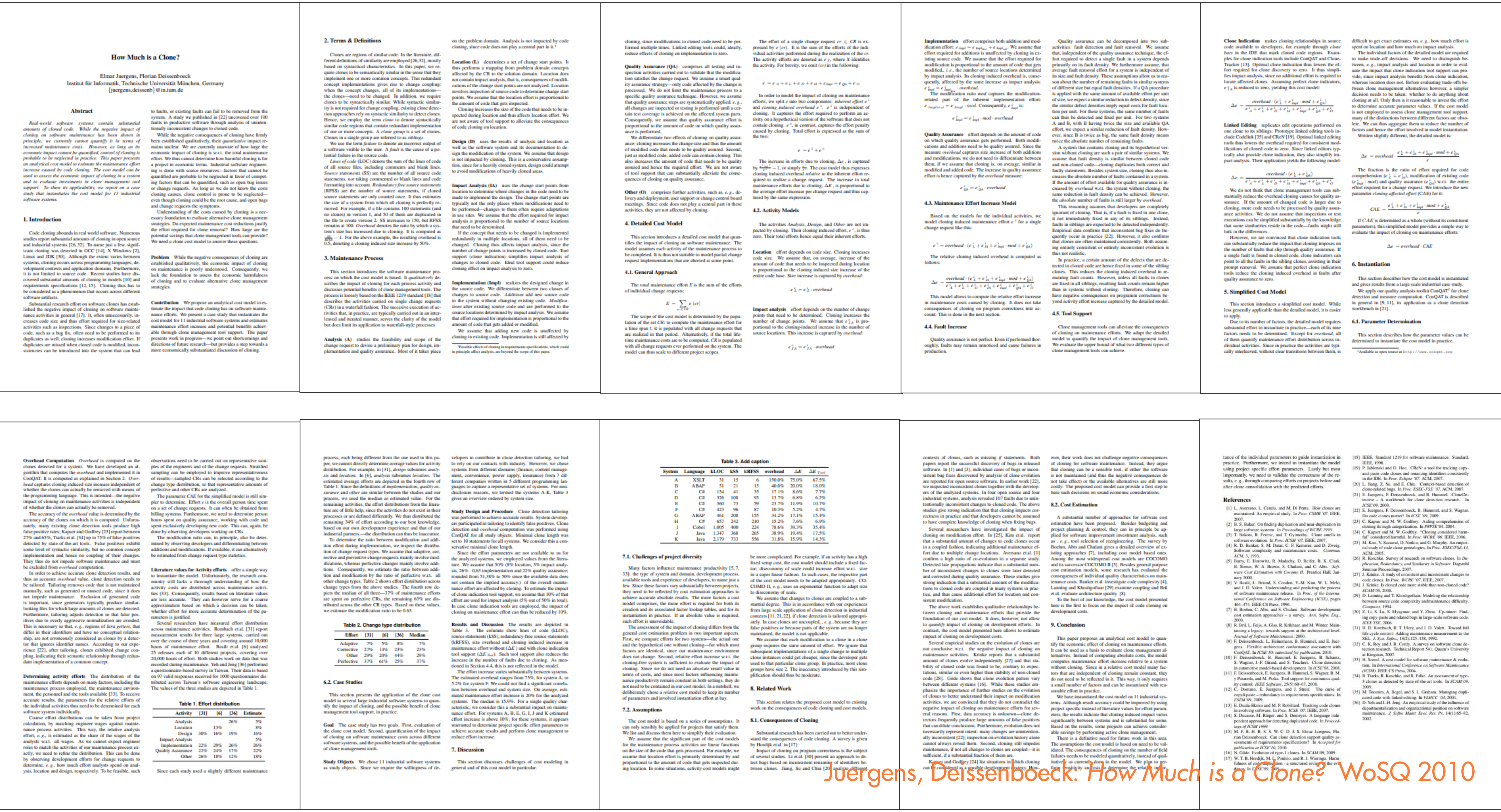
$$165\ \frac{Fehler}{Jahr} \times 3\ \frac{PT}{Fehler}$$

$$495 \frac{PT}{Jahr}$$

$$500 \frac{PT}{Jahr}$$

Munich Re spart durch Einsatz von Clone Management jährlich ca. 500 PT Aufwand für Fehlerbehebung


Komponente A
Komponente B



Juergens, Deissenboeck: *How Much is a Clone?* WoSQ 2010

$$\Delta\text{Aufwand} = \%\text{BlowUp} \times \%\text{CloneAffectedEffort}$$

$$\Delta\text{Aufwand} = \mathbf{\%BlowUp} \times \%\text{CloneAffectedEffort}$$

Blow-Up: 0%

20 Lines

20 Lines

Blow-Up: 50%

20 Lines

20 Lines

20 Lines

$$\Delta\text{Aufwand} = \%\mathbf{12} \times \%\text{CloneAffectedEffort}$$

$$\Delta\text{Aufwand} = \%12 \times \mathbf{\%CloneAffectedEffort}$$

%CloneAffectedEffort

Aktivitäten	Aufwändiger durch Cloning
Analysis	-
Location	Location
Design	-
Impact Analysis	Impact Analysis
Implementation	Implementation
Quality Assurance	Quality Assurance
Other	-

Detaillierte Herleitung und Berechnung im Paper.

Wert für Berechnung: **50%**.

$$\Delta \text{Aufwand} = \%12 \times \mathbf{\%50}$$

$$\Delta \text{Aufwand} = \%12 \times \%50 = \mathbf{6}\%$$

Die Munich Re setzt Clone Detection seit ca. 10 Jahren ein.

Wie sähe es ohne aus?

Continuous Software Quality Control in Practice

Daniela Steidl*, Florian Deissenboeck*, Martin Poehlmann*, Robert Heinke†, Bärbel Uhink-Mergenthaler†
* CQSE GmbH, Garching b. München, Germany
† Munich RE, München, Germany

***Abstract*—Many companies struggle with unexpectedly high maintenance costs for their software development which are often caused by insufficient code quality. Although companies often use static analyses tools, they do not derive consequences from the metric results and, hence, the code quality does not actually improve. We provide an experience report of the quality consulting company CQSE, and show how code quality can be improved in practice: we revise our former expectations on quality control from [1] and propose an enhanced continuous quality control process which requires the combination of metrics, manual action, and a close cooperation between quality engineers, developers, and managers. We show the applicability of our approach with a case study on 41 systems of Munich RE and demonstrate its impact.**

I. Introduction

Software systems evolve over time and are often maintained for decades. Without effective counter measures, the quality of software systems gradually decays [2], [3] and maintenance costs increase. To avoid quality decay, *continuous quality control* is necessary during development and later maintenance [1]: for us, quality control comprises all activities to monitor the system's current quality status and to ensure that the quality meets the quality goal (defined by the principal who outsourced the software development or the development team itself).

Research has proposed various metrics to assess software quality, including structural metrics[1] or code duplication, and has led to a massive development of analysis tools [4]. Much of current research focuses on better metrics and better tools [1], and mature tools such as ConQAT [5], Teamscale [6], or Sonar[2] have been available for several years.

In [1], we briefly illustrated how tools should be combined with manual reviews to improve software quality continuously, see Figure 1: We perceived quality control as a simple, continuous feedback loop in which metric results and manual reviews are used to assess software quality. A quality engineer – a representative of the quality control group – provides feedback to the developers based on the differences between the current and the desired quality. However, we underestimated the amount of required manual action to create an impact. Within five years of experience as software quality consultants in different domains (insurance companies, automotive manufacturers, or engineering companies), we frequently experienced that tool

This work was partially funded by the German Federal Ministry of Education and Research (BMBF), grant EvoCon, 01IS12034A. The responsibility for this article lies with the authors.

[1] e. g., file size, method length, or nesting depth
[2] http://www.sonarqube.org/

Fig. 1. The former understanding of a quality control process

support alone is not sufficient for successful quality control in practice. We have seen that most companies cannot create an impact on their code quality although they employ tools for quality measurements because the pressure to implement new features does not allow time for quality assurance: often, newly introduced tools get attention only for a short period of time, and are then forgotten. Based on our experience, quality control requires actions beyond tool support.

In this paper, we revise our view on quality control from [1] and propose an enhanced quality control process. The enhanced process combines automatic static analyses with a significantly larger amount of manual action than previously assumed to be necessary: Metrics constitute the basis but quality engineers must manually interpret metric results within their context and turn them into actionable refactoring tasks for the developers. We demonstrate the success and practicability of our process with a running case study with Munich RE which contains 32 .NET and 9 SAP systems.

II. Terms And Definitions

- A *quality criterion* comprises a metric and a threshold to evaluate the metric. A criterion can be, *e. g.*, to have a clone coverage below 10% or to have at most 30% code in long methods (*e. g.*, methods with more than 40 LoC).
- *(Quality) Findings* result from a violation of a metric threshold (*e. g.*, a long method) or from the result of a static code analysis (*e. g.*, a code clone).
- *Quality goals* describe the abstract goal of the process and provide a strategy how to deal with new and existing findings during further development: The highest goal is to have no findings at all, *i. e.*, all findings must be removed immediately. Another goal is to avoid new findings, *i. e.*, existing findings are tolerated but new findings must not be introduced. (III-B will provide more information).

III. The Enhanced Quality Control Process

Our quality control process is designed to be *transparent* (all stakeholders involved agree on the goal and consequences

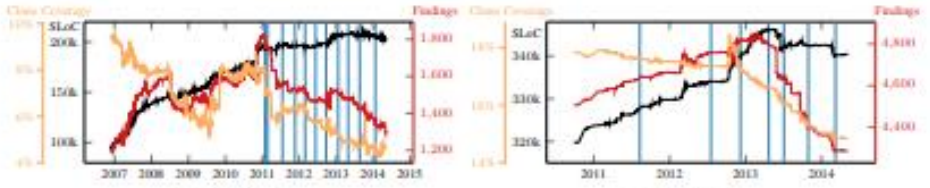


Fig. 3. System *A*

Fig. 5. System *C*

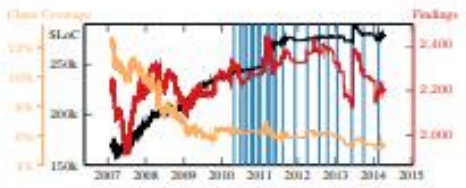


Fig. 4. System *B*

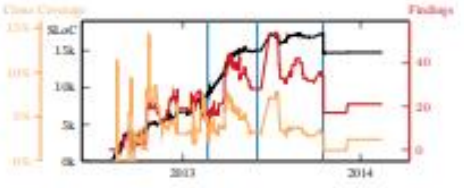


Fig. 6. System *D*

continuously in the available history (Figure 4). The number of findings, however, increases until mid 2012. In 2012, the project switched from QG2 to QG3. After this change, the number of findings decreases and the clone coverage settles around 6%, which is a success of the quality control. The major increase in the number of findings in 2013 is only due to an automated code refactoring introducing braces that led to threshold violations of few hundred methods. After this increase, the number of findings start decreasing again, showing the manual effort of the developers to remove findings.

For System C (Figure 5), the quality control process shows a significant impact after two years: Since the end of 2012, when the project also switched from QG2 to QG3, both the clone coverage and the overall number of findings decline. In the year before, the project transitioned between development teams and, hence, we only wrote two reports (July 2011 and July 2012).

System *D* (Figure 6) almost fulfills QG4 as after 1 year of development, it has only 21 findings in total and a clone coverage of 2.5%. Technically, under QG4, the system should have zero findings. However, in practice, exactly zero findings is not feasible as there are always some findings (*e. g.*, a long method to create UI objects or clones in test code) that are not a major threat to maintainability. Only a human can judge based on manual inspection of the findings whether a system still fulfills QG4, if it does not have exactly zero findings. In the case of System D, we consider 21 findings to be few and minor enough to fulfill QG4.

To summarize, our trends show that our process leads to actual measurable quality improvement. Those trends go beyond anecdotal evidence but are not sufficient to scientifically proof our method. However, Munich RE decided only recently to extend our quality control from the .NET area to all SAP development. As Munich RE develops mainly in the .NET and SAP area, most application development is now supported by quality control. The decision to extend the scope of quality control confirms that Munich Re is convinced by the benefit of quality control. Since the process has been established, maintainability issues like code cloning are now an integral part of discussions among developers and management.

V. Conclusion

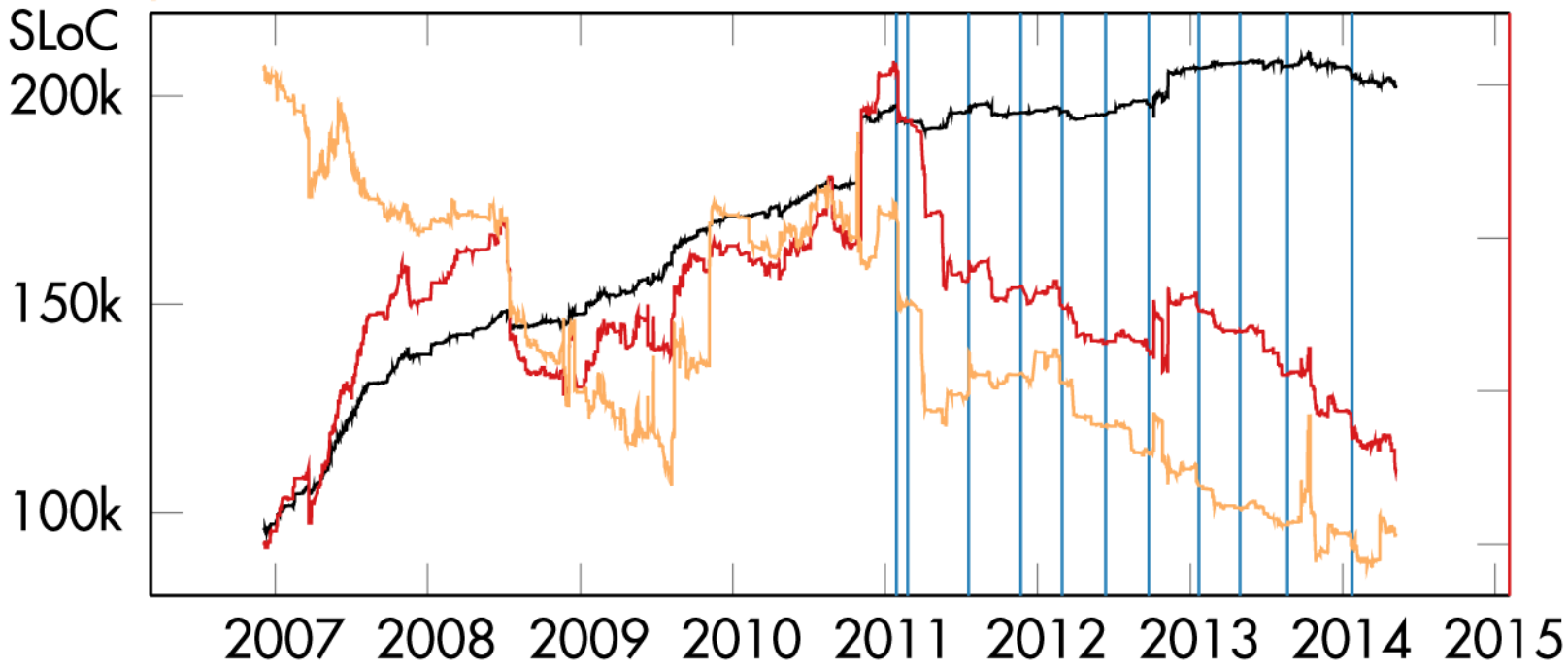
Quality analyses must not be solely based on automated measurements, but need to be combined with a significant amount of human evaluation and interaction. Based on our experience, we proposed a new quality control process for which we provided a running case study of 41 industry projects. With a qualitative impact analysis at Munich RE we showed measurable, long-term quality improvements. Our process has led to measurable quality improvement and an increased maintenance awareness up to management level at Munich Re.

References

[1] F. Deissenboeck, E. Juergens, B. Hummel, S. Wagner, B. M. y Parareda, and M. Pizka, "Tool support for continuous quality control," in *IEEE Software*, 2008.
[2] D. L. Parnas, "Software aging," in *ICSE '94*.
[3] S. G. Eick, T. L. Graves, A. F. Karr, J. S. Marron, and A. Mockus, "Does code decay? Assessing the evidence from change management data," *IEEE Trans. Software Eng.*, 2001.
[4] P. Johnson, "Requirement and design trade-offs in hackystat: An in-process software engineering measurement and analysis system," in *ESEM'07*.
[5] F. Deissenboeck, M. Pizka, and T. Seifert, "Tool support for continuous quality assessment," in *STEP'05*.
[6] L. Heinemann, B. Hummel, and D. Steidl, "Teamscale: Software quality control in real-time," in *ICSE'14*.
[7] R. C. Martin, *Clean Code: A Handbook of Agile Software Craftsmanship*, 2008.

Steidl, Heinke, Uhink: Continuous Software Quality Control in Practice? *ICSME* 2014

Einsparung durch Clone Detection



Menge an geklontem Code hat sich seit der Einführung von Clone Detection halbiert.
Ohne Clone Detection wäre der Clone Blow-Up daher vorr. doppelt so groß.

Ersparnis Aufwand = **6%**

Munich Re spart durch Einsatz von Clone Detection jährlich 6% Aufwand durch vermiedene Redundanz ein.

Findings

All	**6793**
Architecture	**1**
Architecture Conformance	1
Code Anomalies	**1344**
Bad practice	971
Correctness	2
Exception Handling	62
General checks (built-in)	120
Null pointer dereference	13
Performance	36
Unused code	93
Unused variable or parameter	47
Code Duplication	**988**
Cloning	101
Redundant Literals	887
Documentation	**3378**
Comment completeness	3236
Task tags	142
Formatting	**6**
Code formatting	6
Naming	**110**
Java naming conventions	110
Structure	**966**
File Size	38
Method Length	278
Nesting Depth	650

$$500 \frac{PT}{Jahr}$$

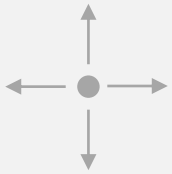
Munich Re spart durch Einsatz von Clone Management jährlich ca. 500 PT Aufwand für Fehlerbehebung

$$\text{Ersparnis Aufwand} = \mathbf{6}\%$$

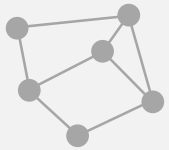
Munich Re spart durch Einsatz von Clone Detection jährlich 6% Aufwand durch vermiedene Redundanz ein.

Best Practices

Probleme **differenziert** betrachten



Kontext berücksichtigen



Experten involvieren

Konservative Annahmen machen

Visualisierungen gezielt einsetzen

Schnelles Feedback minimiert Kosten

Dr. Nils Göde · goede@cqse.eu · +49 176 10452662
Dr. Elmar Jürgens · juergens@cqse.eu · +49 179 675 3863

CQSE GmbH
Centa-Hafenbrädl-Straße 59
81249 München