

Was bringt die Behebung von technischen Schulden?

Ergebnisse einer retrospektiven Nutzenrechnung nach 10 Jahren



Requirement
Engineers



User

2006 2007 2008 2009 2010 2011 2012 2013 2014 2015 2016 2017 2018 2019 2020

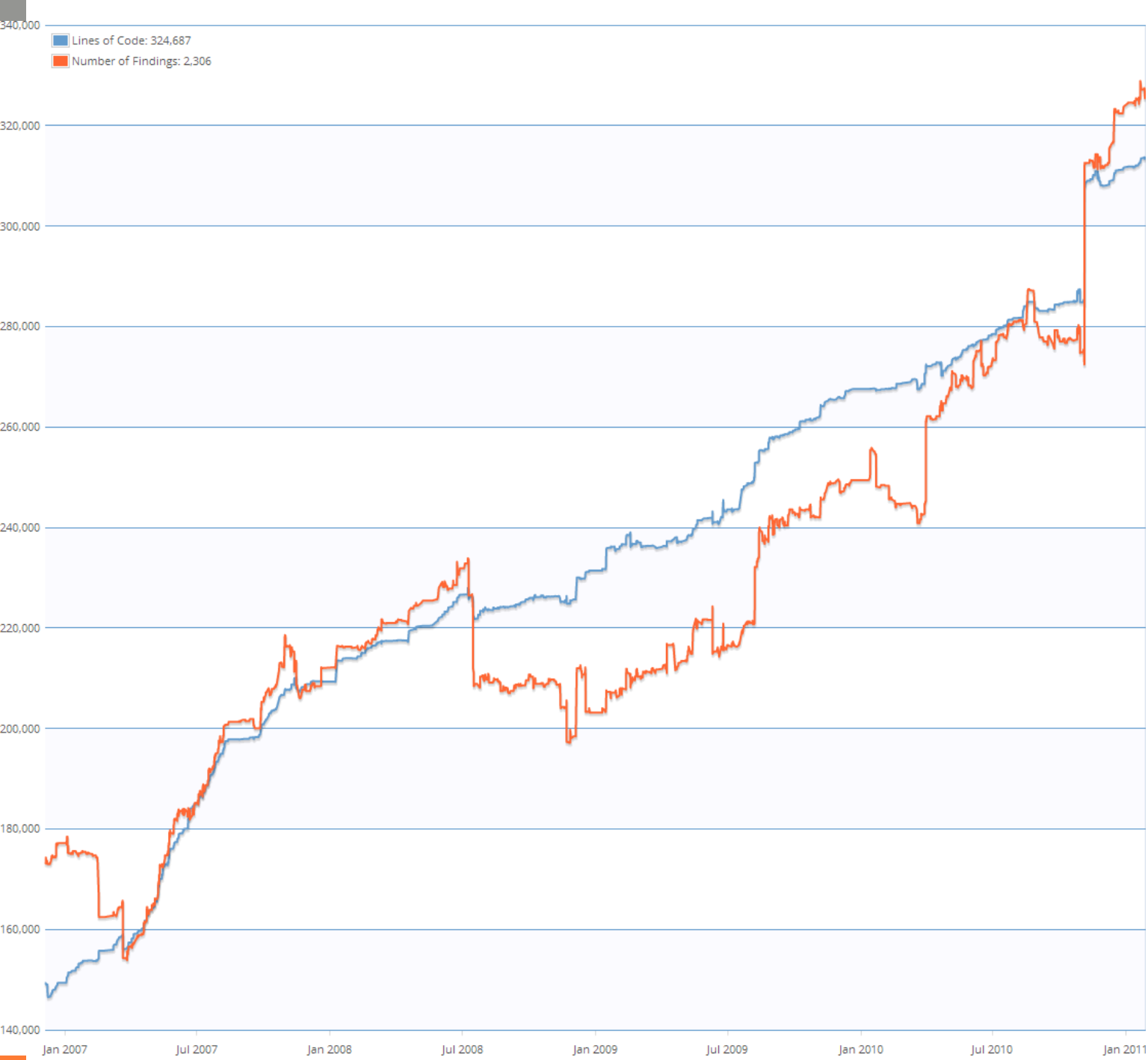


TreeAdministrationQuarterly.cs

```
93  /// </summary>
94  protected override bool DoLazyLoading(
95      UltraTreeNode node)
96  {
97      // Some types of segments always have the
98      // expand icon
99      if ( node.Tag is GaSegment ||
100         node.Tag is MainSegment ||
101         node.Tag is GeneralMainSegment ||
102         node.Tag is Branch )
103         return true;
104     // All others only if there are child nodes
105     return false;
106 }
107
108 /// <summary>
109 /// Structure segments return only loss or
110 /// premium segments according to
111 /// the mode.
112 /// </summary>
113 protected override IList GetChildSegments(
114     ISegment parent, ref bool alreadySorted )
115 {
116     if ( parent is StructureSegment )
117     {
118         // Show either premium or loss beyond
119         // structure segment
120         StructureSegment strucSeg = (StructureSegment
121             )parent;
122         if ( (TypeOfSgmtEnum) GetFilterValue( typeof(
123             TypeOfSgmtEnum) ) == TypeOfSgmtEnum.Loss
124             )
125             return strucSeg.GetLossProcessingSegments();
126         else
127             return strucSeg.GetPremiumProcessingSegments
128             ();
129     }
130 }
```

TreeAdministrationYearly.cs

```
106  /// </summary>
107  protected override bool DoLazyLoading(
108      UltraTreeNode node)
109  {
110      // Some types of segments always have the
111      // expand icon
112      if ( node.Tag is GaSegment ||
113         node.Tag is MainSegment ||
114         node.Tag is GeneralMainSegment ||
115         node.Tag is Branch )
116         return true;
117     // All others only if there are child nodes
118     return false;
119 }
120
121 /// <summary>
122 /// Structure segments return only loss or
123 /// premium segments according to
124 /// the mode.
125 /// </summary>
126 protected override IList GetChildSegments(
127     ISegment parent, ref bool alreadySorted )
128 {
129     if ( parent is StructureSegment )
130     {
131         // Show either premium or loss beyond
132         // structure segment
133         StructureSegment strucSeg = (StructureSegment
134             )parent;
135         if ( (TypeOfSgmtEnum) GetFilterValue( typeof(
136             TypeOfSgmtEnum) ) == TypeOfSgmtEnum.Loss
137             )
138             return strucSeg.GetLossProcessingSegments();
139         else
140             return strucSeg.GetPremiumProcessingSegments
141             ();
142     }
143 }
```

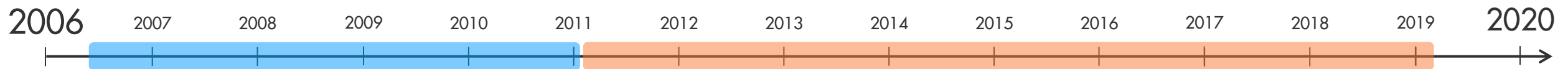




Requirement Engineers



User

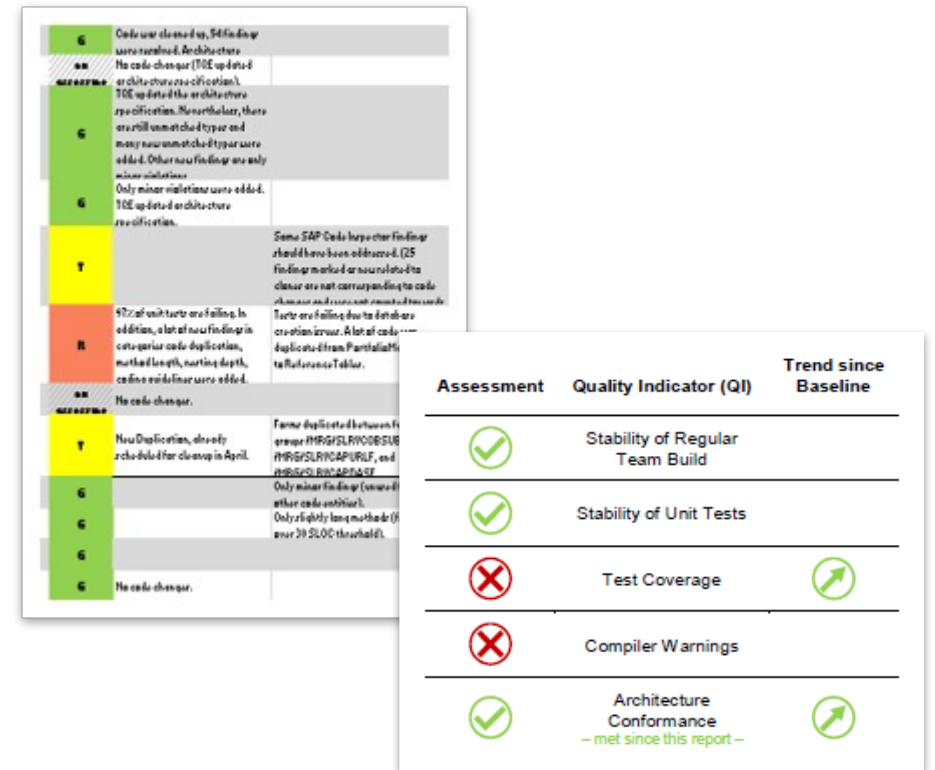


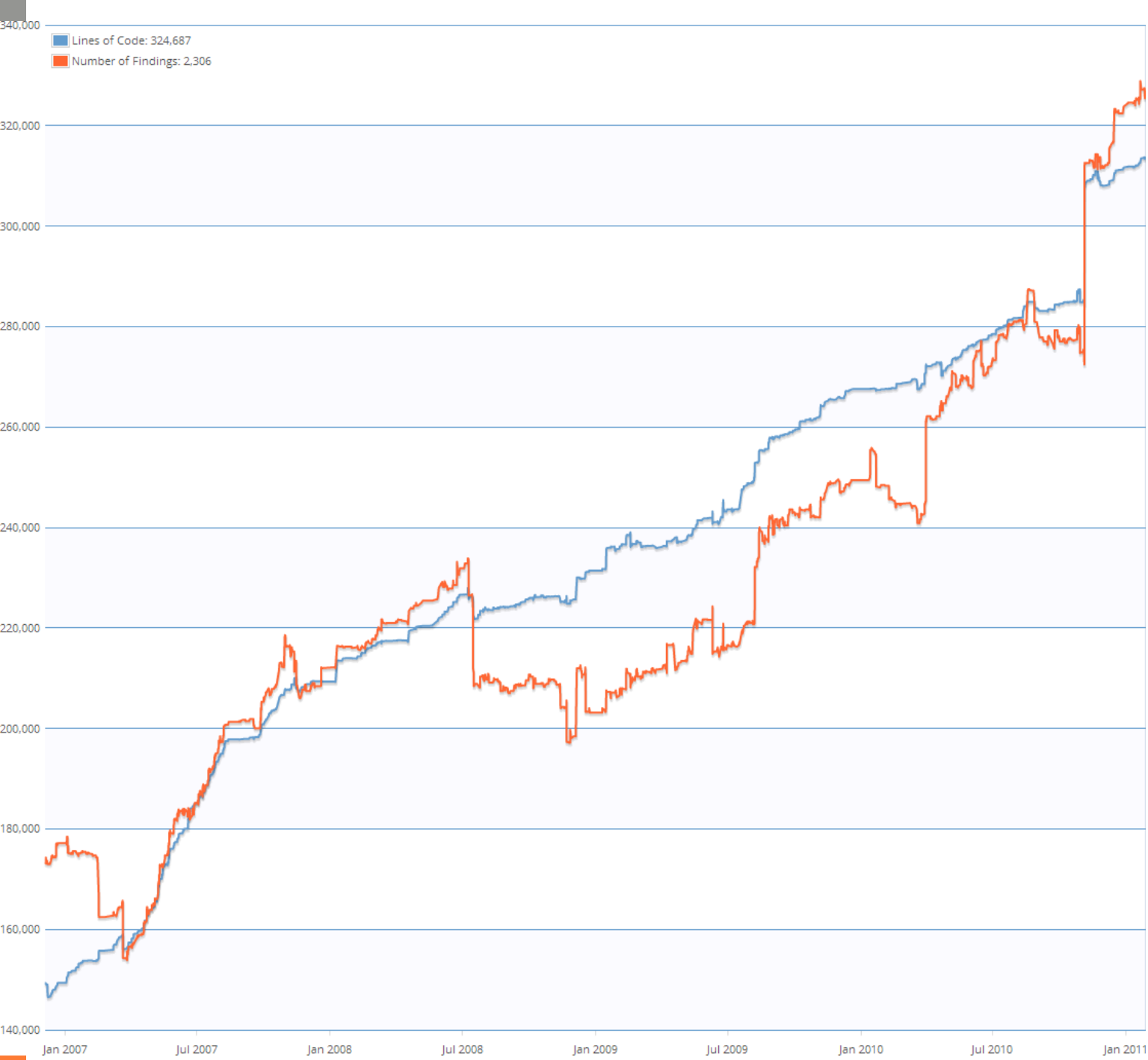
Munich Re Internal Services

Dashboards, IDE Plugin, Azure DevOps



Monthly Assessments, Reports





Project	QG	Assessment	Comment	Detailed Comment	QG-relevant Findings
[REDACTED]	2	Y	Four new security relevant findings should have been reviewed.	Missing authorization checks (1) at the beginning of report [REDACTED] GOODSRECEIPT. (2) before CALL TRANSACTION ycl_p2p_invoice=>gc_ta_f90 in method create_post_values in class [REDACTED] FIXEDASSET (3) before CALL TRANSACTION gc_ta_fbd1 in method start_recurring_entry in class [REDACTED] INVOICE	4



Di 29.10.2019 18:01

[REDACTED] Munich-MR

AW: Security Code Scan - Monthly Assessment [REDACTED]

To IT TQE-TGA (Pool) - Munich-MR; [REDACTED]

Cc Proft Uwe - Munich-MR

You replied to this message on 30.10.2019 15:06.

Hi all,

the findings down below has been fixed, next assessment status should be green again.

Best regards

[REDACTED]

Wie können wir den
Nutzen quantifizieren?

```
// 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();
}
```

```
// 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();
}
```

```
// 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();
}
```

```
// 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();
}
```

Do Code Clones Matter?

Elmar Jaergens, Florian Deissenboeck, Benjamin Hummel, Stefan Wagner
Institut für Informatik, Technische Universität München
Boltzmannstr. 3, 85748 Garching b. München, Germany
[jaergens,deissenboeck,hummel,wagner]@tum.de

Abstract

Code cloning is not only assumed to inflate maintenance costs but also identified to propagate an accumulation of faults. This paper presents the results of a large-scale case study that was undertaken to find out if (1) clones are changed independently, (2) if these inconsistencies are introduced in clusters, (3) if unintentional inconsistencies can be represented faults. In this case study we analyzed three commercial systems written in C#, one written in Cobol and one open-source system written in Java. To conduct the study we developed a new detection algorithm that enables us to detect inconsistent clones. We manually inspected about 900 clone groups to handle the inevitable false positives and discarded each of the over 700 inconsistent clone groups with the developers of the respective systems to determine if the inconsistencies are intentional and if they represent faults. Altogether, around 1800 individual clone group assessments were manually performed in the course of the case study. The study lead to the identification of 107 faults that have been confirmed by the systems' developers.

1. Clones & correctness

Research in software maintenance has shown that many programs contain a significant amount of duplicated (cloned) code. Such cloned code is considered harmful for two reasons: (1) multiple, possibly unnecessary, duplicate of code increase maintenance costs and, (2) unnecessary changes to cloned code can create faults and, hence, lead to incorrect program behavior [19, 28]. While clone detection has been a very active area of research in recent years up to now, there is no thorough understanding of the degree of harmfulness of code cloning. In fact, some researchers even started to doubt the harmfulness of cloning at all [16]. This paper, therefore, we investigate the effects of code cloning on program correctness. It is important to understand, that clones do not directly cause faults but inconsistent changes to clones can lead to incorrect program behavior. A particularly dangerous type of change to cloned code is the *inconsistent bug fix*. If a fault was

purposes, the more complicated average performance would be more adequate. Thus, and to assess the complexity of the entire pipeline we executed the detection on the source code of Eclipse¹, limiting detection to a certain amount of code. Our results on an Intel Core 2 Duo 2.4 GHz running Java in a single thread with 3.5 GB of RAM are shown in Figure 5. The settings are the same as for the main study (min clone length of 10, min edit distance of 5). It is capable to handle the 5.6 MLOC of Eclipse in about 3 hours, which is fast enough to be executed within a nightly build.

5. Study description

In order to gain a solid insight into the effects of inconsistent clones, we use a study design with 3 objects and 3 research questions that guide the investigation.

5.1. Study objects

We chose 2 objects and 1 open source project as sources of software systems. This resulted in 3 analyzed projects in total. We chose systems written in different languages, by different teams in different companies and with different functionalities to increase the transferability of the study results. These objects included 3 systems written in C#, a Java system as well as a long-lived Cobol system. All these systems are already in production. For non-disclosure reasons we chose the commercial systems names from A to D. An overview is shown in Table 1.

Munich Re Group The Munich Re Group is one of the largest insurance companies in the world. This not only means that we can find them at all but also that they are considered a significant part of the real world of a system. It does not make sense to analyze inconsistent clones if they are a pure phenomenon.

LV 1871 The Lebensversicherung von 1871 a.G. is a German health insurance company. The system has 171 developers and maintains several custom software systems for maintenance and PCs. In this study, we analyze 5 custom-developed and manually constructed clones written in Cobol/2 that employ about 150 users.

ES The Eclipse IDE is a well-known open source system in the world of Eclipse IDEs with 3.1 million users.

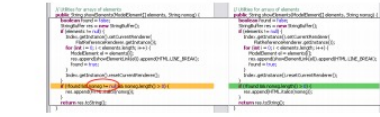


Figure 5. Missing null check on right side can cause exception (Sylphius).

2. Terms and definitions

For a thorough discussion of the consequences of inconsistent clones, we define that a *fault* is an incorrect output of a software visible to the user and that a *clone* is the cause of a potential failure inside the code. *Defects* are the subset of faults and failures.

3. Related work

A substantial amount of research has been dedicated to code cloning in recent years. The detailed survey by Koschik [19] or Roy and Corby [24] provide a comprehensive overview of existing work. Since this paper targets consequences of cloning and detection of inconsistencies, we detail existing work in these areas.

17 44 46

Kritisch Nutzersichtbar

17 44 46

Kritisch Nutzersichtbar

17 44 46

Kritisch Nutzersichtbar

17 44 46

Kritisch Nutzersichtbar

17 44 46

Kritisch Nutzersichtbar

17 44 46

a small fraction of inconsistent clones becomes consistent again through later changes, potentially inducing a larger degree of independence of clones than hitherto believed. Geiger et al. [9] report that a relation between change counts and code clones could, contrary to expectations, not be statistically verified. Lotz and Werningering [20] report that no systematic relationship between code cloning and changeability could be established.

• Instead of manual inspection of the actual inconsistent clones to evaluate consequences for maintenance and correctness, indirect measures¹ are used [1, 9, 22, 23, 28, 27]. Such approaches are inherently inaccurate and can easily lead to misleading results. For example, unintentional differences and faults, while unknown to developers, exhibit the same evolution pattern as intentional independent evolution and are thus prone to misclassification.

• The analyzed systems are too small to be representative [17] or consist of industrial software [1, 2, 9, 17, 22, 26].

• The analyses specifically focus on faults introduced during creation [12, 25] or evolution [2] of clones, i.e., habiting quantification of inconsistencies in general.

17 44 46

Kritisch Nutzersichtbar

17 44 46

Kritisch Nutzersichtbar

17 44 46

Kritisch Nutzersichtbar

17 44 46

Kritisch Nutzersichtbar

17 44 46

Kritisch Nutzersichtbar

17 44 46

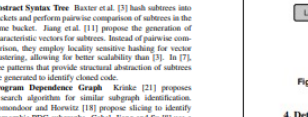


Figure 2. The clone detection pipeline used

4. Detecting inconsistent clones

This section explains the approach used for detecting inconsistent clones in large amounts of code. Our approach of finding clones at the level, which usually is sufficient for finding copy-pasted code, while at the same time being efficient. The algorithm works by constructing a suffix tree of the code and then for each possible suffix an approximate search based on the edit distance in this tree is performed.

Our clone detector is organized as a pipeline, which is depicted in Figure 2. The files under analysis are loaded and then fragmented by the analyzer, yielding a stream of tokens, which is filtered to exclude comments and generated code (recognized by user provided patterns). From the token stream, which consist of single keywords, identifiers, operators, and so on, the *normalizer* normalizes statements. This stage performs normalization, such that

17 44 46

Kritisch Nutzersichtbar

17 44 46

Kritisch Nutzersichtbar

17 44 46

Kritisch Nutzersichtbar

17 44 46

Kritisch Nutzersichtbar

17 44 46

Kritisch Nutzersichtbar

17 44 46

17 44 46

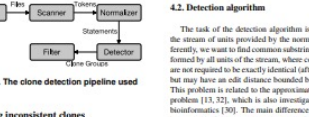


Figure 4. Different UV behavior since right side does not use operations (Sylphius).

This section explains the approach used for detecting inconsistent clones in large amounts of code. Our approach of finding clones at the level, which usually is sufficient for finding copy-pasted code, while at the same time being efficient. The algorithm works by constructing a suffix tree of the code and then for each possible suffix an approximate search based on the edit distance in this tree is performed.

Our clone detector is organized as a pipeline, which is depicted in Figure 2. The files under analysis are loaded and then fragmented by the analyzer, yielding a stream of tokens, which is filtered to exclude comments and generated code (recognized by user provided patterns). From the token stream, which consist of single keywords, identifiers, operators, and so on, the *normalizer* normalizes statements. This stage performs normalization, such that

17 44 46

Kritisch Nutzersichtbar

17 44 46

Kritisch Nutzersichtbar

17 44 46

Kritisch Nutzersichtbar

17 44 46

Kritisch Nutzersichtbar

17 44 46

Kritisch Nutzersichtbar

17 44 46

17 44 46

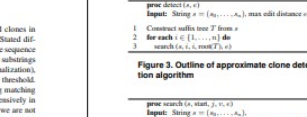


Figure 3. Outline of approximate clone detection algorithm

This section explains the approach used for detecting inconsistent clones in large amounts of code. Our approach of finding clones at the level, which usually is sufficient for finding copy-pasted code, while at the same time being efficient. The algorithm works by constructing a suffix tree of the code and then for each possible suffix an approximate search based on the edit distance in this tree is performed.

Our clone detector is organized as a pipeline, which is depicted in Figure 2. The files under analysis are loaded and then fragmented by the analyzer, yielding a stream of tokens, which is filtered to exclude comments and generated code (recognized by user provided patterns). From the token stream, which consist of single keywords, identifiers, operators, and so on, the *normalizer* normalizes statements. This stage performs normalization, such that

17 44 46

Kritisch Nutzersichtbar

17 44 46

Kritisch Nutzersichtbar

17 44 46

Kritisch Nutzersichtbar

17 44 46

Kritisch Nutzersichtbar

17 44 46

Kritisch Nutzersichtbar

17 44 46

17 44 46

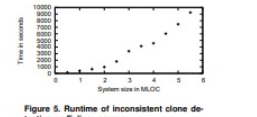


Figure 6. Runtime of inconsistent clone detection on Eclipse source

This section explains the approach used for detecting inconsistent clones in large amounts of code. Our approach of finding clones at the level, which usually is sufficient for finding copy-pasted code, while at the same time being efficient. The algorithm works by constructing a suffix tree of the code and then for each possible suffix an approximate search based on the edit distance in this tree is performed.

Our clone detector is organized as a pipeline, which is depicted in Figure 2. The files under analysis are loaded and then fragmented by the analyzer, yielding a stream of tokens, which is filtered to exclude comments and generated code (recognized by user provided patterns). From the token stream, which consist of single keywords, identifiers, operators, and so on, the *normalizer* normalizes statements. This stage performs normalization, such that

17 44 46

Kritisch Nutzersichtbar

17 44 46

Kritisch Nutzersichtbar

17 44 46

Kritisch Nutzersichtbar

17 44 46

Kritisch Nutzersichtbar

17 44 46

Kritisch Nutzersichtbar

17 44 46

17 44 46

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

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

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

#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
- $8,25 \text{ Millionen SLOC} / 50k = 165$

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

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

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

Ø Fehlerfolgekosten 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

Ø Fehlerfolgekosten 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

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

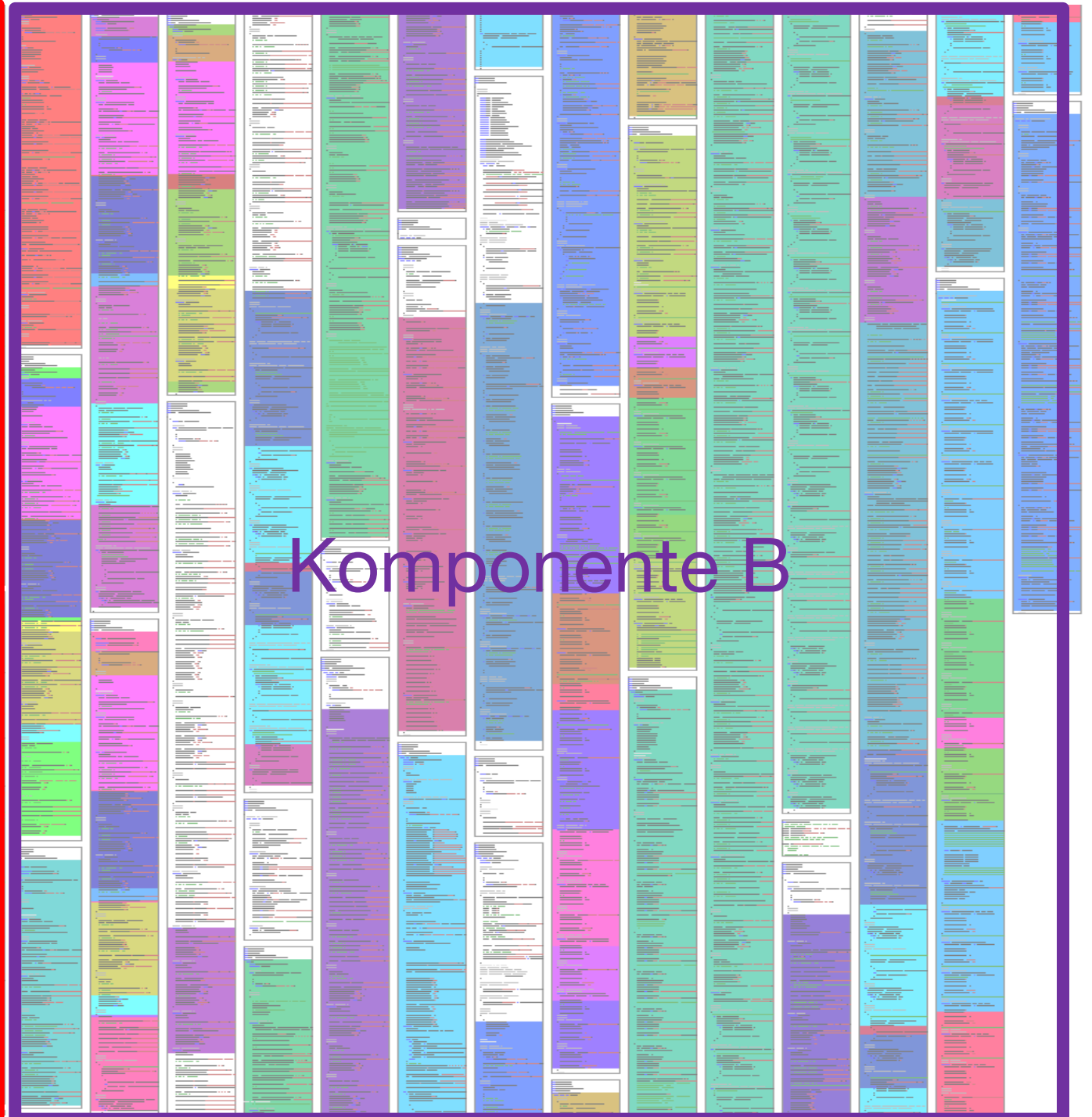
$$165 \frac{\text{Fehler}}{\text{Jahr}} \times 3 \frac{\text{PT}}{\text{Fehler}}$$

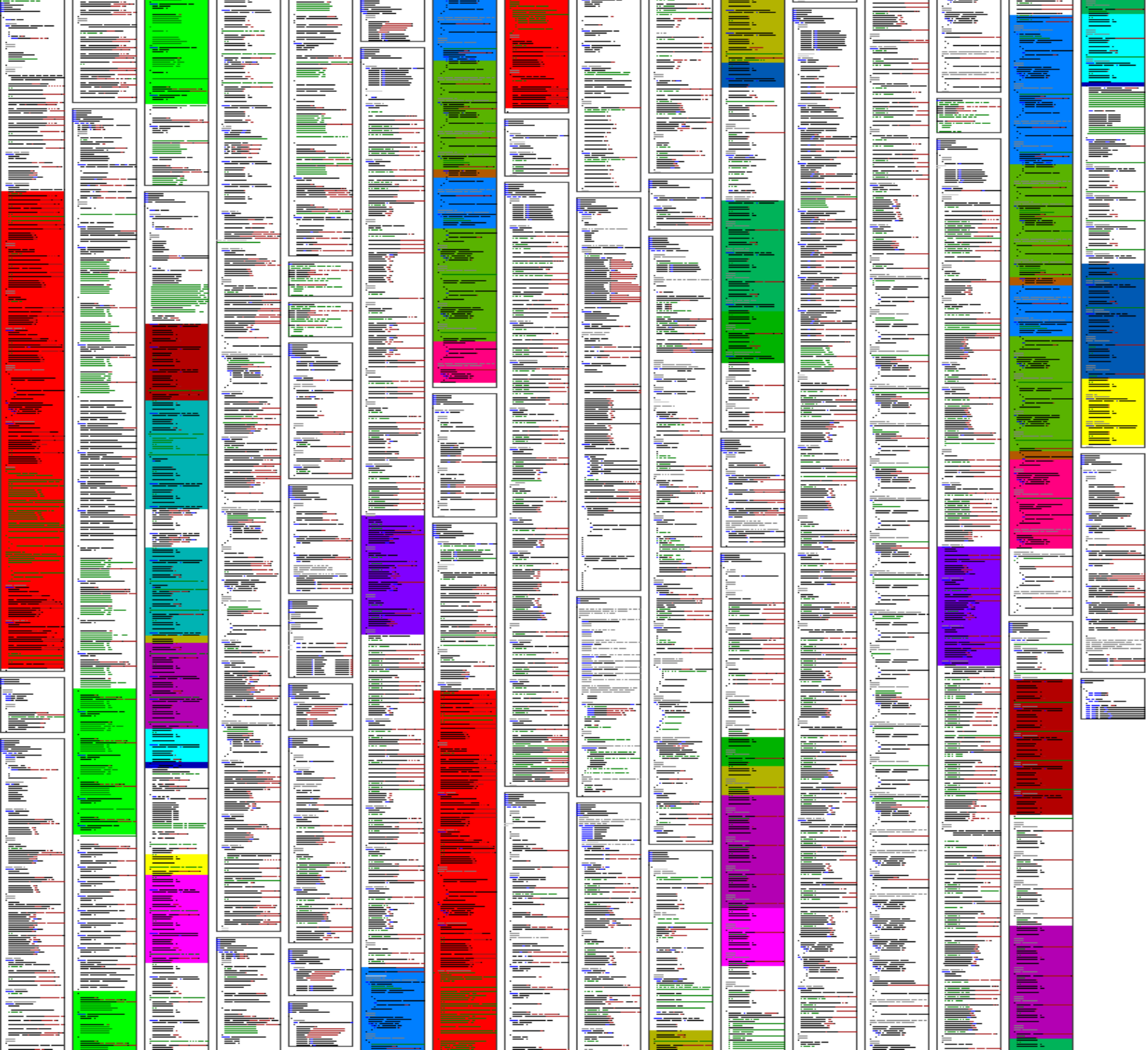
495 $\frac{PT}{Jahr}$

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

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







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

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

Blow-Up: 0%



20 Lines

20 Lines

Blow-Up: 50%



20 Lines

20 Lines

20 Lines

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

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

%CloneAffectedEffort

Aktivitäten

- Analysis
- Location
- Design
- Impact Analysis
- Implementation
- Quality Assurance
- Other

Aufwändiger durch Cloning

-
- Location
-
- Impact Analysis
- Implementation
- Quality Assurance
-

Detaillierte Herleitung und Berechnung im Paper.

Wert für Berechnung: **50%**.

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

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

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 Pöhlmann*, 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 metrics¹ 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 Sonar² 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



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

must client specify either party, earning less's but

QSE site. operators

bone, the seeds, safety

sting findings fly).

side – about meant

costs ways (7)).

findings

under

ologies coming front.

metrics code length, culture lines

metrics, and

However, cooperation in 2006, in average had e decreases

ize the history once, although Teamscale to

safety (see

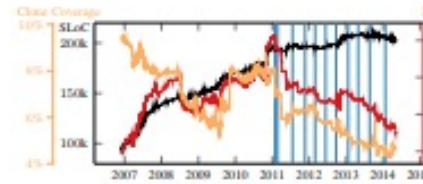


Fig. 3. System A

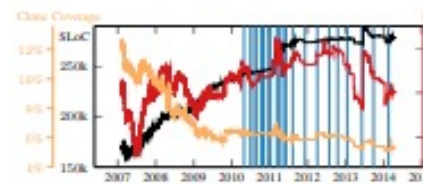


Fig. 4. System B

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 prove our method. However, Munich RE decided only recently to extend our quality control from the .NET area to all SAP

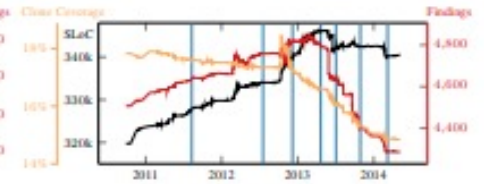


Fig. 5. System C

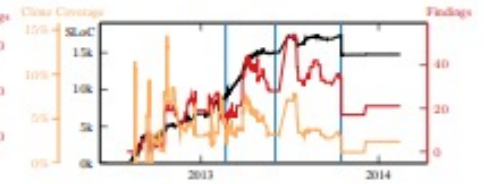


Fig. 6. System D

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

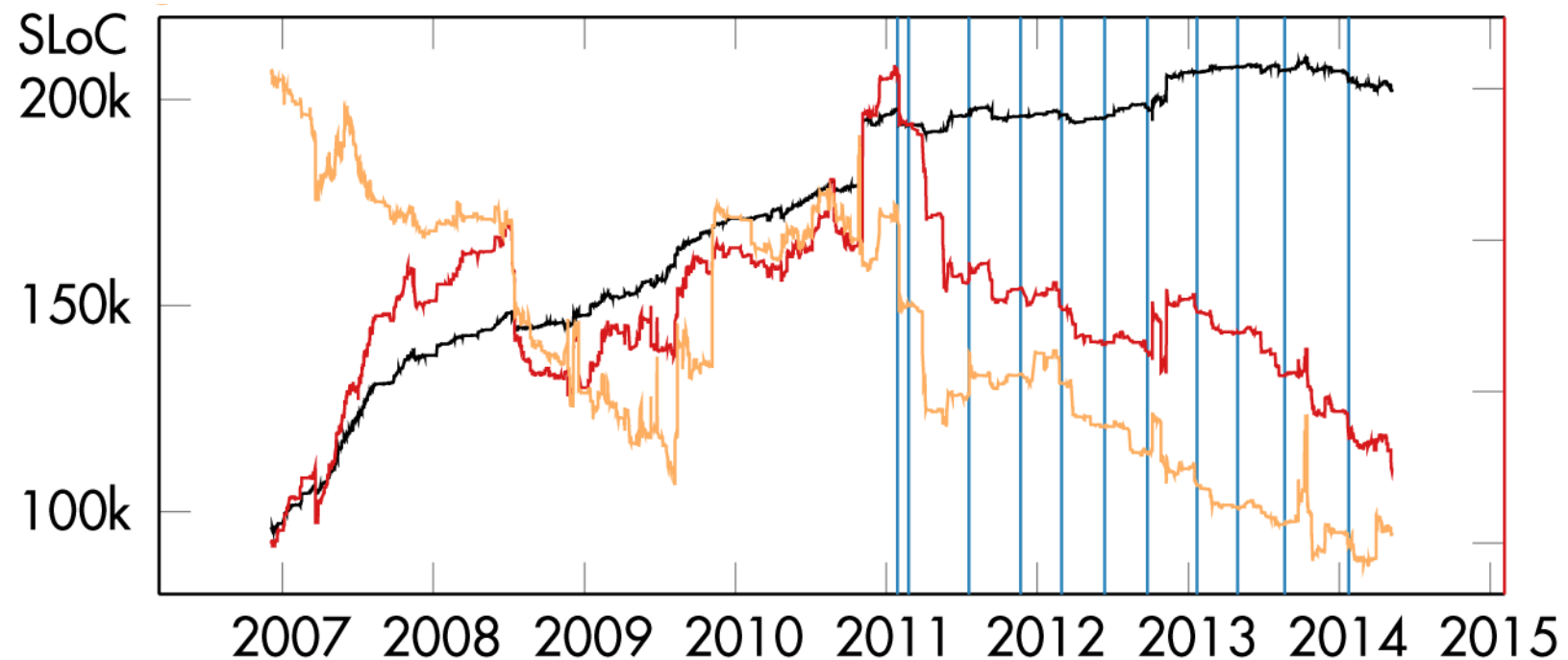
- [1] F. Deissenboeck, E. Juergens, B. Hummel, S. Wagner, B. M. y Parareda, and M. Pirka, "Tool support for continuous quality control," in *IEEE Software*, 2008.
- [2] D. L. Parnas, "Software aging," in *ICSE '94*.
- [3] S. G. Eick, T. L. Graves, A. F. Karr, J. S. Marron, and A. Mockus, "Does code decay? Assessing the evidence from change management data," *IEEE Trans. Software Eng.*, 2001.
- [4] P. Johnson, "Requirement and design trade-offs in backstat: An in-process software engineering measurement and analysis system," in *ESEM'07*.
- [5] F. Deissenboeck, M. Pirka, 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.

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

¹e.g., file size, method length, or nesting depth

²<http://www.sonarqube.org/>

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

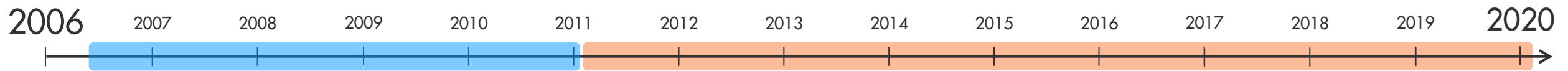
<input checked="" type="checkbox"/> All	6793
<input checked="" type="checkbox"/> Architecture	1
<input checked="" type="checkbox"/> Architecture Conformance	1
<input checked="" type="checkbox"/> Code Anomalies	1344
<input checked="" type="checkbox"/> Bad practice	971
<input checked="" type="checkbox"/>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

500 $\frac{PT}{Jahr}$

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

Ersparnis Aufwand = 6%

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



Kontakt – Ich freue mich auf Fragen 😊



Dr. Elmar Jürgens

juergens@cqse.eu

+49 179 675 3863

CQSE GmbH
Centa-Hafenbrädl-Str 59
81249 München
www.cqse.eu

The logo for CQSE, featuring the letters 'CQSE' in a bold, sans-serif font. The 'Q' is colored orange, while the 'C', 'S', and 'E' are in grey.