Rückblick
Link zu Online-Artikel

LABORJOURNAL LABORJOURNAL Blog

Start Wissen Methoden & mehr Stellen Meinung Termine Spaß Archiv Service Mediadaten

Warum klemmt's bei Open Science und Co.?

(07.06.2022) Das Implementierungsdefizit von Open Science und der Reform der Forschungsbewertung haben auch die Wissenschaftler selbst zu verantworten.

tweet teilen mitteilen pin it teilen teilen

Im November vergangenen Jahres haben die 193 Mitgliedstaaten der UNESCO, also alle von der UN anerkannten unabhängigen Länder dieser Erde, „Empfehlungen zur Offenen Wissenschaft (Open Science)" unterzeichnet. Darin verpflichten sich die Staaten, eine Kultur der offenen Wissenschaft zu fördern, in diese zu investieren und Anreize für sie zu schaffen. Dies war wohl nur ein vor-

Implementierungsdefizit von Open Science
- Gründe
- Möglichkeiten, diesem Defizit entgegenzuwirken
- …

Dirnagl (2022)

Rückblick
Link zu Online-Artikel

Natürlich gäbe es viele weitere Möglichkeiten. Von keiner wissen wir, wie praktikabel sie ist, ob sie tatsächlich ihren Zweck erfüllt oder eher gar nicht-intendierte, negative Auswirkungen hat. Aber nur in der Anwendung finden wir das heraus, durch Pilotprojekte und maßvolle Modifikation bestehender Verfahren. Was funktioniert, wird ausgebaut – was nicht funktioniert, wird verbessert oder aufgegeben.

Und im Idealfall sollten solche Pilotprojekte gleich als „Interventionen" verstanden und wissenschaftlich begleitet werden. Schließlich kann Evaluations- beziehungsweise Implementierungsforschung solide Evidenz liefern, um Institutionen, Fördergeber und Wissenschaftler bei der Umsetzung von spezifischen Maßnahmen rational zu beraten.

Die Fördergeber sollten all dies unterstützen, indem sie für solche Maßnahmen und Pilotprojekte Förderlinien ausschreiben – aber auch für die begleitende Implementierungsforschung. Auch sollten sie die Beantragung von Mitteln ermöglichen, die Open Science fördern. Also zum Beispiel für Forschungsdatenmanagement, wissenschaftliches Qualitätsmanagement, Data Stewards, Patienten- und Stakeholder-Engagement und so weiter.

Last but not least müssen auch die Geldgeber der Universitäten aktiv werden. Sie sollten die Berechnung der Landeszuführungsbeiträge auch an die Umsetzung von offener Wissenschaft knüpfen – so wie sie dies ja auch schon für Open Access und Gleichstellung mit einigem Erfolg gemacht haben.

Ob das alles dann weniger braucht als zwei Dekaden, wage ich zu bezweifeln. Aber das sollte uns nicht abhalten, jetzt aktiv zu werden. Gut Ding will eben Weile haben.

Ulrich Dirnagl

Sehr spannend wurde es zum Ende des Artikels…

Maßnahmen:
- Ausprobieren
- Ausselektieren
- Verbessern
- Braucht Zeit
- Beginnt trotzdem

Dirnagl (2022)

Source Code Review for Scientific Issues: Version 1.0

Kathrin Fucke

Hintergrund
Fehler in Software Quellcode

- ∅ **3 Fehler** pro 1000 Zeilen Software-Quellcode **EXPERTEN!**
- Große Projekte: mehrere Millionen Zeilen Codes
- Google-Quellcode: über 2 Milliarden Zeilen

Ariane 5: teuerster Bug

- Unbemannte Trägerrakete, gestartet am 04.06.1996
- Zerstörung wenige Sekunden nach Start
- Kettenreaktion ausgelöst durch **eine Zeile Quellcode**

Grotelüschen (2008); JeiceTheWarrior (2010); Llaguno, Source, & Manager (2017); Schilling (2015)

Hintergrund
Fehler in Software Quellcode

Auszug aus meinem Quellcode in R

```
# creates a data frame with descriptive and inference statistic data of physical attractiveness ratings
  for(i in 1 : length(database_levels)) # interates databases
  {
    for(j in 1 : length(gender_levels)) # interates target gender
    {
      data_subset <- dplyr::filter(data, Database_shortcut == database_levels[i] & GenderSelf == gender_levels[j]) # filters database and gender
      data_meta[nrow(data_meta) + 1, ] <- c(database_levels[i], gender_levels[j], describe(data_subset$Attractive)) # adds row for each database and gender.
      data_meta$nv_pvalue[nrow(data_meta)] <- shapiro.test(data_subset$Attractive)$p.value
      one_sample_t_test <- t.test(data_subset$Attractive, mu = data_subset$Scale_midpoint[1])
      data_meta$t_test_statistic[nrow(data_meta)] <- one_sample_t_test$statistic
      data_meta$t_test_pvalue[nrow(data_meta)] <- one_sample_t_test$p.value #adds p-value one sample t-test
      data_meta$t_test_confint_LL[nrow(data_meta)] <- one_sample_t_test$conf.int[1] #adds confidence limits (lower bound)
      data_meta$t_test_confint_UL[nrow(data_meta)] <- one_sample_t_test$conf.int[2] #adds confidence limits (upper bound)
      data_meta$d[nrow(data_meta)] <- MBESS::ci.sm(Mean = data_meta$mean[nrow(data_meta)] - data_subset$Scale_midpoint[1],
                                                   SD = data_meta$sd[nrow(data_meta)], N = data_meta$n[nrow(data_meta)])$Standardized.Mean #adds effect size
      data_meta$r[nrow(data_meta)] <- effectsize::d_to_r(d = as.double(data_meta$d[nrow(data_meta)])) # converts effect size d to r
        }
     }
```

SCHWÄCHEN

Source Code Review = manuelle Inspektion des Quellcodes ohne Daten

Abgrenzung: Testen = mit Daten

Ziele

Warum?

- Prozessbegleitende interne Qualitätssicherung
- Entlastung des Peer-Reviews
- Eigene Arbeit transparent machen
- Willigkeit für Open Science
- …

Inhalt: gegenständlich Protokoll (zum Abheften, öffentlich machen) mit…

- Festlegung von Ziel und Inhalt des Reviews
- Verantwortlichkeiten und Fristen
- Unterschrift
- Nachweis der Ergebnisse

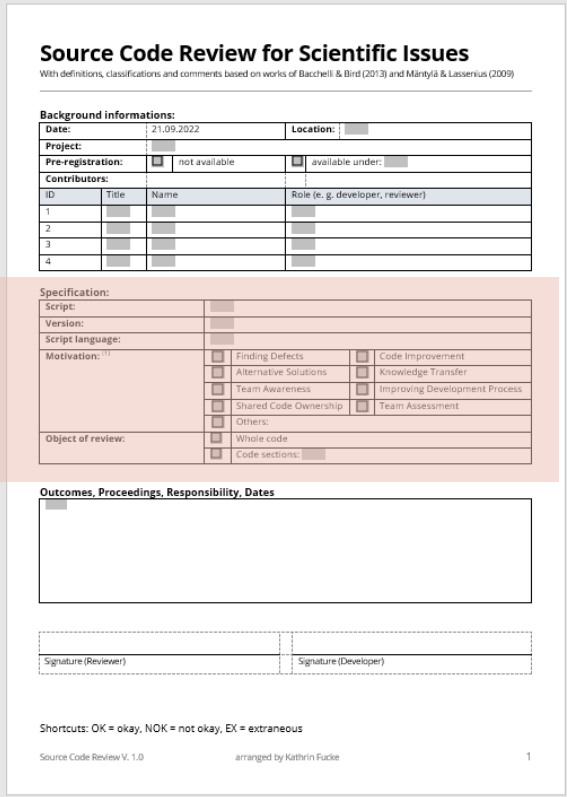
Source Code Review for Scientific Issues
Version 1.0

Basis:
- Bacchelli & Bird (2013): Motive des Reviews
- Mäntylä & Lassenius (2009): Klassifikation von Defekten

Ergebnis:
- 5-seitiges **Word Dokument**
- Schreibgeschützt
- Kein Passwort → Schreibschutz kann für Veränderungen aufgehoben werden

Ihr könnt: verändern, ergänzen, streichen…

Seite 1



Source Code Review for Scientific Issues

With definitions, classifications and comments based on works of Bacchelli & Bird (2013) and Mäntylä & Lassenius (2009)

Background informations:

Date:	21.09.2022	**Location:**	
Project:			
Pre-registration:	not available	available under:	
Contributors:			

ID	Title	Name	Role (e. g. developer, reviewer)
1			
2			
3			
4			

Specification:

Script:				
Version:				
Script language:				
Motivation: [1]		Finding Defects		Code Improvement
		Alternative Solutions		Knowledge Transfer
		Team Awareness		Improving Development Process
		Shared Code Ownership		Team Assessment
		Others:		
Object of review:		Whole code		
		Code sections:		

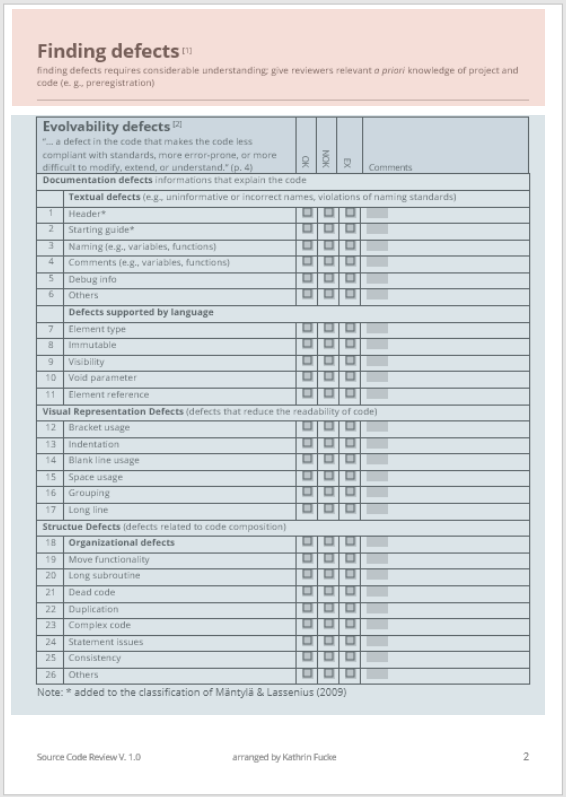
Outcomes, Proceedings, Responsibility, Dates

Signature (Reviewer)	Signature (Developer)

Shortcuts: OK = okay, NOK = not okay, EX = extraneous

Source Code Review V. 1.0 arranged by Kathrin Fucke 1

Seite 2



Finding defects [1]

finding defects requires considerable understanding; give reviewers relevant *a priori* knowledge of project and code (e. g., preregistration)

Evolvability defects [2]

"... a defect in the code that makes the code less compliant with standards, more error-prone, or more difficult to modify, extend, or understand." (p. 4)

		OK	NOK	EX	Comments
Documentation defects informations that explain the code					
	Textual defects (e.g., uninformative or incorrect names, violations of naming standards)				
1	Header*				
2	Starting guide*				
3	Naming (e.g., variables, functions)				
4	Comments (e.g., variables, functions)				
5	Debug info				
6	Others				
	Defects supported by language				
7	Element type				
8	Immutable				
9	Visibility				
10	Void parameter				
11	Element reference				
Visual Representation Defects (defects that reduce the readability of code)					
12	Bracket usage				
13	Indentation				
14	Blank line usage				
15	Space usage				
16	Grouping				
17	Long line				
Structue Defects (defects related to code composition)					
18	**Organizational defects**				
19	Move functionality				
20	Long subroutine				
21	Dead code				
22	Duplication				
23	Complex code				
24	Statement issues				
25	Consistency				
26	Others				

Note: * added to the classification of Mäntylä & Lassenius (2009)

Source Code Review V. 1.0 arranged by Kathrin Fucke 2

Source Code Review
Exkurs: Motivation, Ergebnis

Motivation
Hauptmotiv: Defekte finden

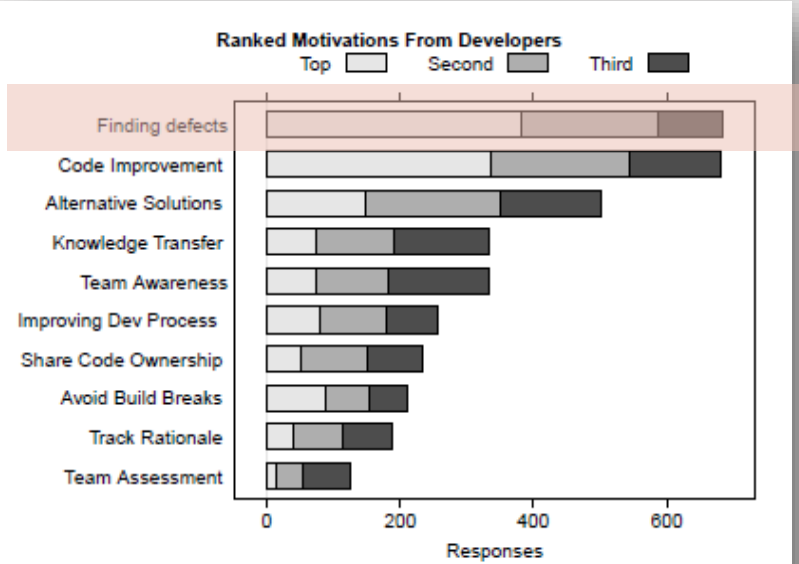


Fig. 3. Developers' motivations for code review.

„Gap“

Ergebnis
Hauptergebnis: Codeverbesserung

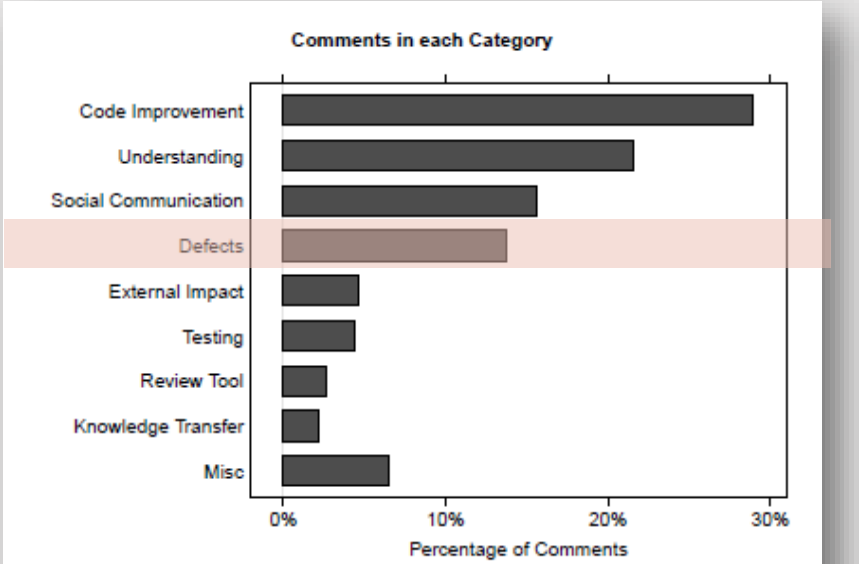


Fig. 4. Proportion of comments by card sort category.

Bacchelli & Bird (2013)

Source Code Review

Grund für Gap

- Identifikation von Fehler erfordert **hohes Maß an Verständnis**
- Um Code zu verstehen und Defekte zu finden - brauchen Reviewer Unterstützung
- durch eine **exzellente Dokumentation und Verständlichkeit des Codes** …
- „Open“ reicht nicht aus

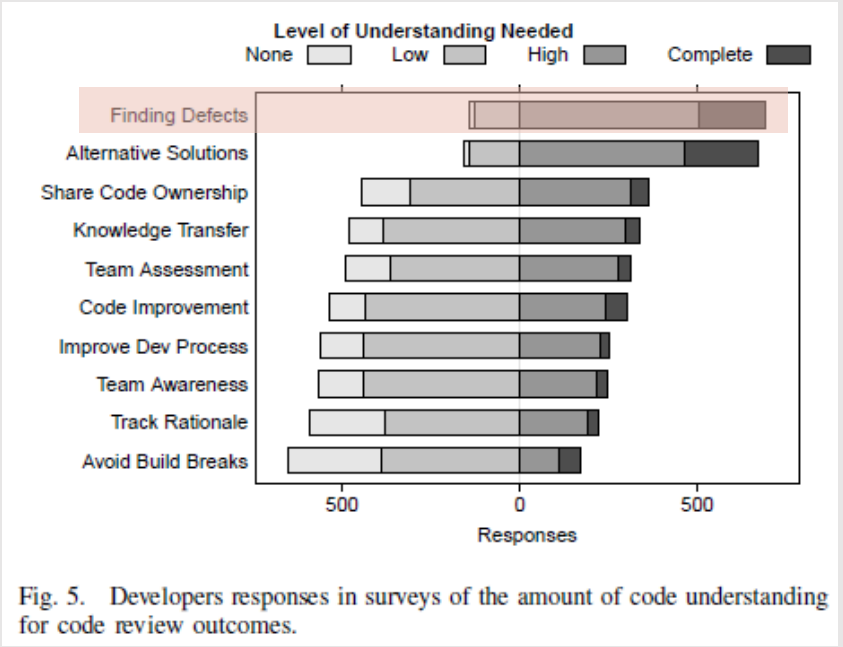


Fig. 5. Developers responses in surveys of the amount of code understanding for code review outcomes.

Bacchelli & Bird (2013)

Source Code Review
Klassifikation Defekte

Evolvability Defects (71.1% | 77.4%)

führen dazu, dass der Code schlechter verständlich, nicht in Übereinstimmung mit Standards oder schwierig zu adaptieren ist

Functional Defects (21.4% | 13.2 %)

können Systemfehler verursachen

False Positives (7.5% | 9.4 %)

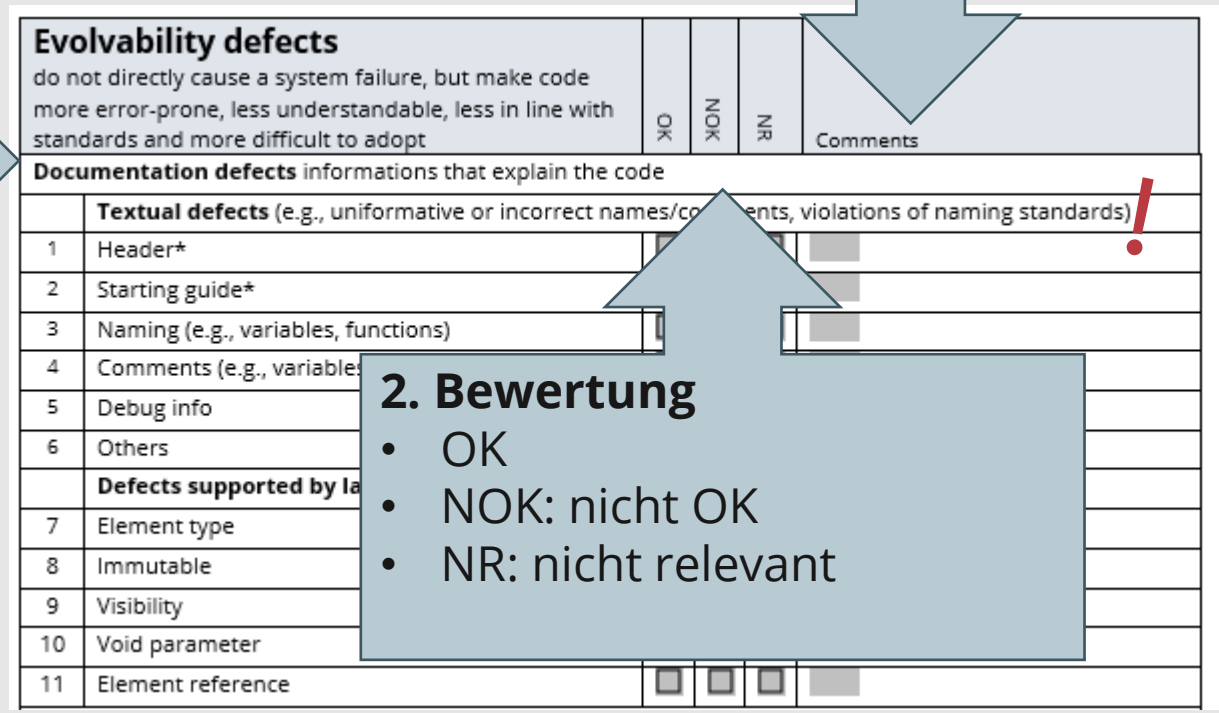
Defekte identifiziert durch industrielle | studentische Reviewer

Mäntylä & Lassenius (2009)

Source Code Review
Klassifikation Defekte

1. Defektkategorie und Kriterien

3. Kommentare

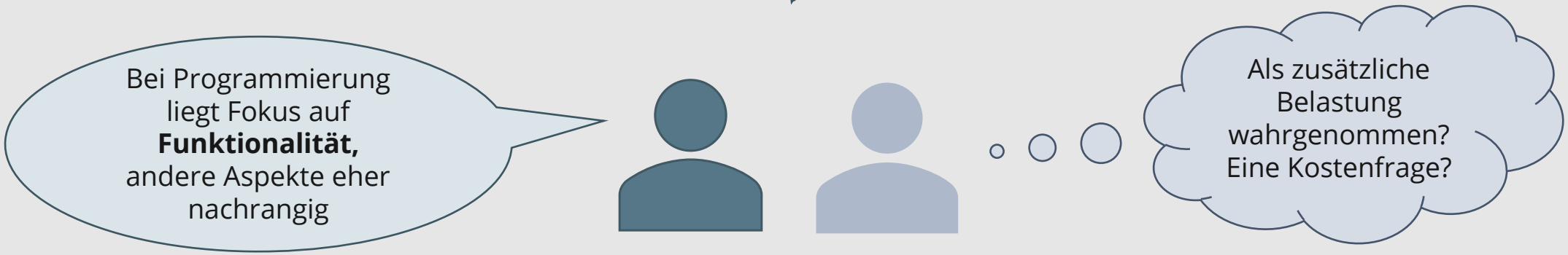


Evolvability defects do not directly cause a system failure, but make code more error-prone, less understandable, less in line with standards and more difficult to adopt		OK	NOK	NR	Comments
Documentation defects informations that explain the code					
	Textual defects (e.g., uniformative or incorrect names/co ents, violations of naming standards)				
1	Header*	☐	☐	☐	
2	Starting guide*	☐	☐	☐	
3	Naming (e.g., variables, functions)	☐	☐	☐	
4	Comments (e.g., variable				
5	Debug info				
6	Others				
	Defects supported by la				
7	Element type				
8	Immutable				
9	Visibility				
10	Void parameter				
11	Element reference	☐	☐	☐	

2. Bewertung
- OK
- NOK: nicht OK
- NR: nicht relevant

Source Code Review
Herausforderungen

Problem I: QM-Maßnahmen sind **nicht produktiv**



Wie kann man sich QM-Maßnahmen in der Softwareprogrammierung leisten?

- Optimal: 50% der Programmierkosten, realistisch: 10% - 20%
- Kosten in Projektantrag aufnehmen, ggf. als optionale Kosten
- Flexibilität: Kosten bei neuen Projekten höher…
- Verantwortung auch in der Hände der Geldgeber, systematisch (Un-)Willigkeit beobachten

Source Code Review
Herausforderungen

- Problem II: Standards
- Welche Standards werden für das Source Code Review genutzt? Z. B. Was ist eine Long Line?
- Gibt es Konventionen für Open Methods?
- Teamkonvention?

Fazit

- „Open“ reicht nicht aus.
- Der Source Code muss auch kontrolliert werden.
- Dafür muss Source Code Qualitätskriterien entsprechen und verständlich sein.
- Fokus besonders auch auf Evolvability Defects, z. B. Dokumentation.
- QM gibt es nicht zum Null-Tarif.
- Kosten sollten eingeplant werden.

Referenzen

Bacchelli, A., & Bird, C. (2013). Expectations, outcomes, and challenges of modern code review. *Proceedings of the International Conference on Software Engineering (ICSE)*, 712–721. https://doi.org/10.1109/ICSE.2013.6606617

Dirnagl, U. (2022, 07. Juni). *Warum klemmt's bei Open Science und Co.?*. Laborjournal. https://www.laborjournal.de/editorials/2516.php?s=03

Grotelüschen, F. (2008, 29. Juli). *Der Absturz der Ariane 5*. Deutschlandfunk. https://www.deutschlandfunk.de/der-absturz-der-ariane-100.html

JeiceTheWarrior. (2010, 21. September). *Longer video of 'Ariane 5' Rocket first launch failure/explosion* [Video]. YouTube. https://www.youtube.com/watch?v=gp_D8r-2hwk&t=1s

Llaguno, M., Source, O., & Manager, S. (2017). 2017 Coverity Scan Report. Open Source Software—The Road Ahead. In *Synopsys*.

Mäntylä, M. V., & Lassenius, C. (2009). What types of defects are really discovered in code reviews? *IEEE Transactions on Software Engineering*, *35*(3), 430–448. https://doi.org/10.1109/TSE.2008.71

Referenzen

Schilling, A. (2015, 21. September). *Codebasis von Google umfasst 2 Milliarden Zeilen und 86 TB*. xxhardwareLUXX. https://www.hardwareluxx.de/index.php/news/allgemein/wirtschaft/36640-codebasis-von-google-umfasst-2-milliarden-zeilen-und-86-tb.html