2020 SimTIC

UNICEP
RIO CLARO

Code Review como Garantia de Qualidade em Projetos Ágeis

Prof. Dr. Erik Aceiro Antonio – aceiro@gmail.com

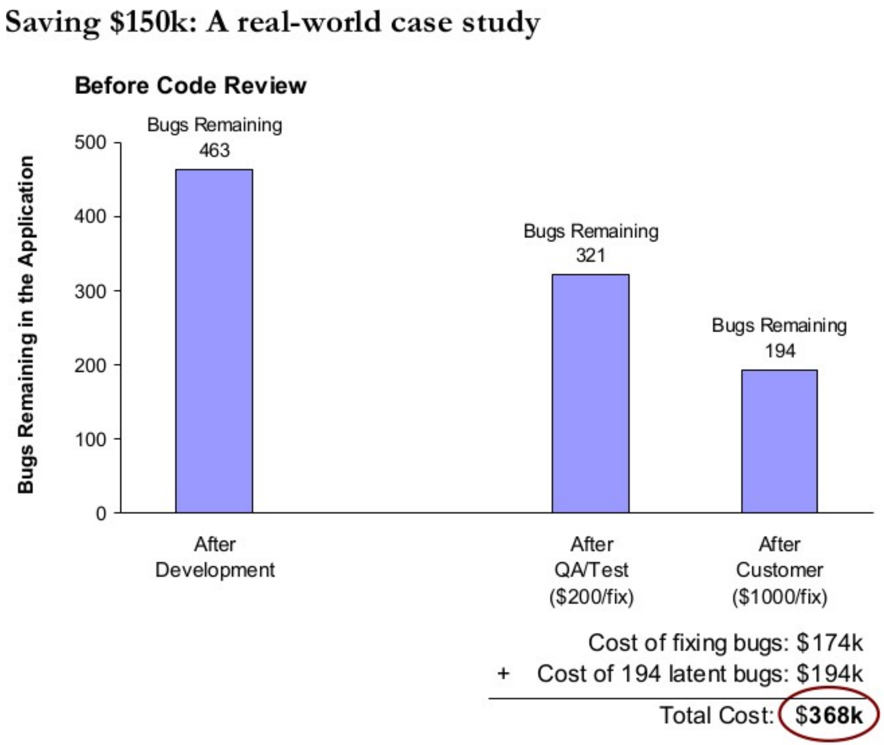
- Doutorado em Engenharia de Software – UFSCar/SP
- Mestre em Engenharia Elétrica – Mackenzie/SP
- Graduação em Ciência da Computação – Mackenzie/SP

Coordenador Sistemas de Informação – UNICEP

Code Review como Garantia de Qualidade em Projetos Ágeis

- Sumário
  - Qualidade de Software
  - Atividades de Teste
    - **Visão Geral de Code Review**
  - Modern Code Review
  - Ferramentas para Code Review
  - Exemplo

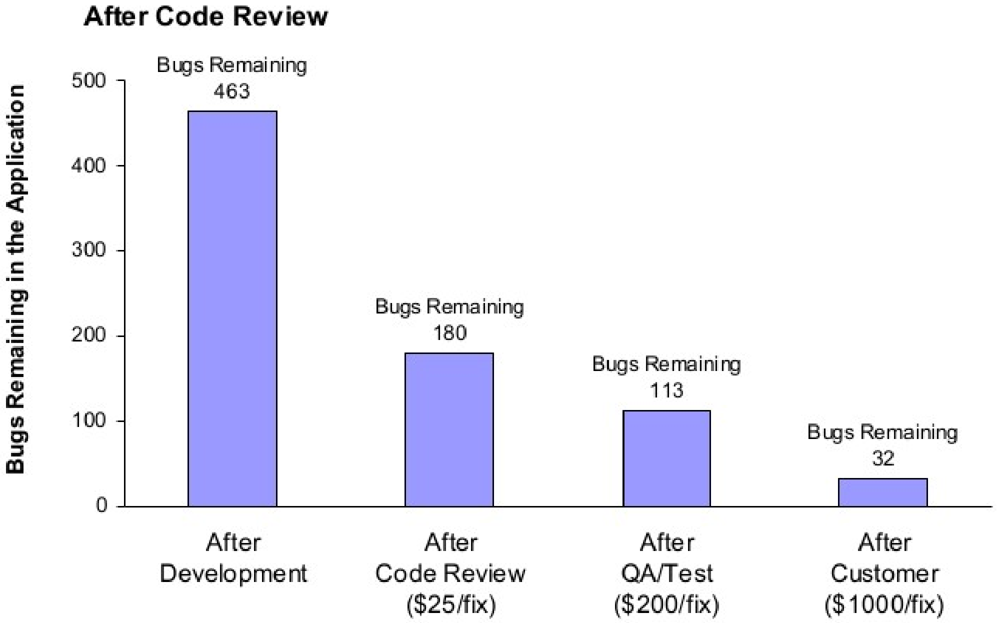
Qualidade de Software



Saving $150k: A real-world case study

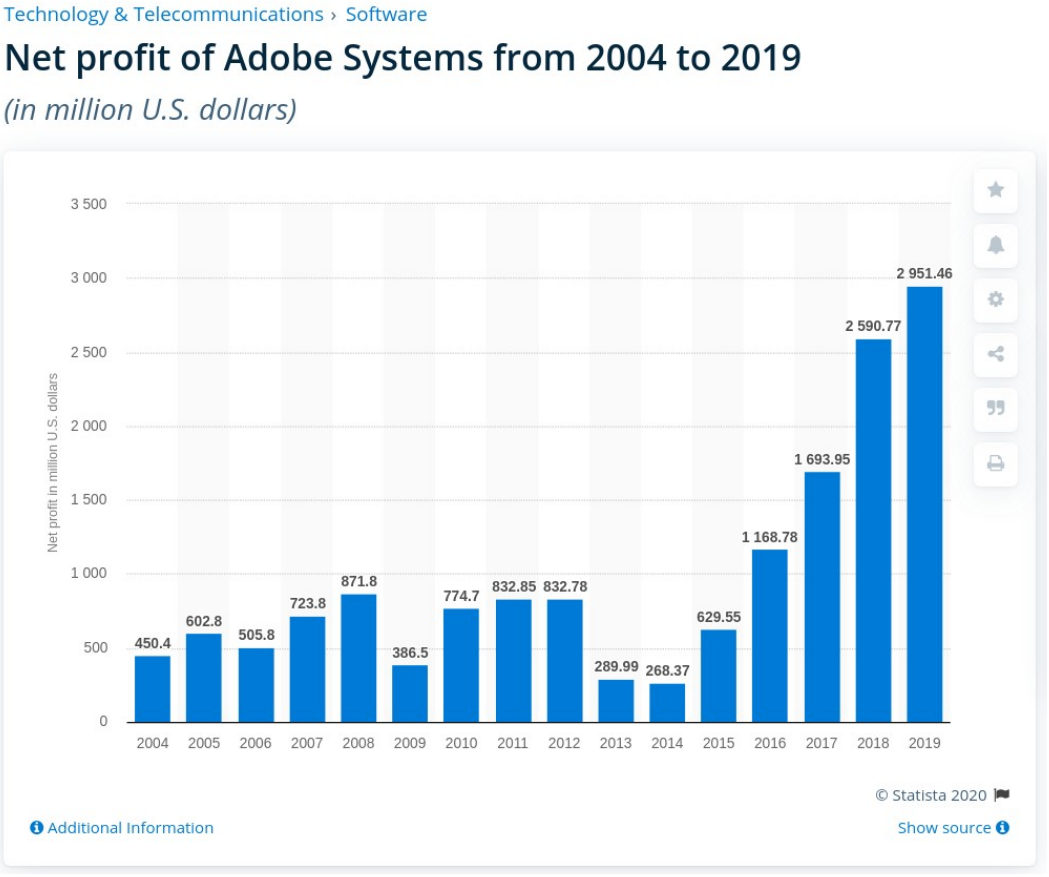
Cost of fixing bugs: $174k
+ Cost of 194 latent bugs: $194k
Total Cost: **$368k**

Qualidade de Software



	Cost of fixing bugs:	$120k
+	Cost of 32 latent bugs:	$ 32k
	Total Cost:	$**152k**

Bug de 1 Bilhão de dolares



Technology & Telecommunications › Software

Net profit of Adobe Systems from 2004 to 2019

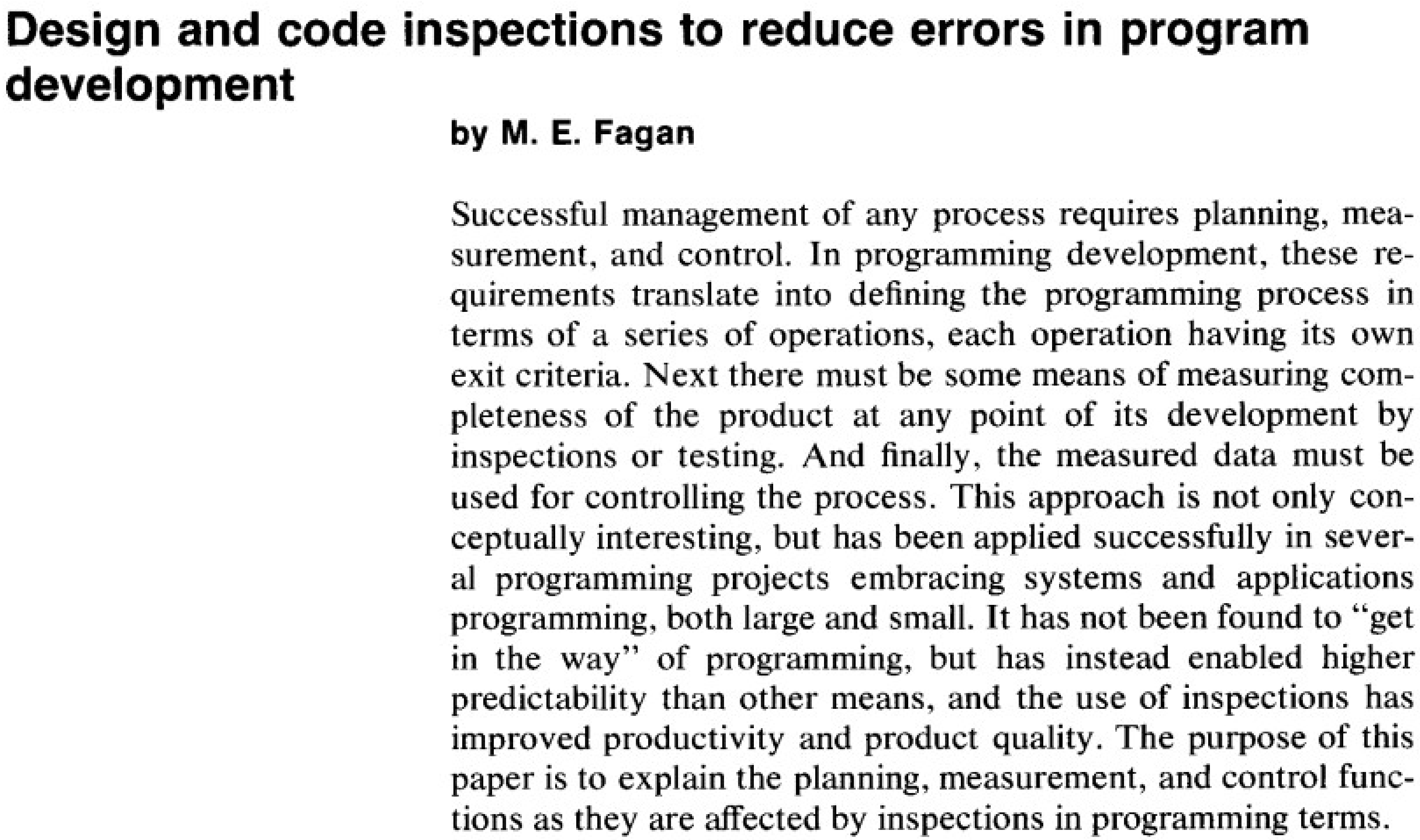
(in million U.S. dollars)

© Statista 2020

Additional Information

Show source

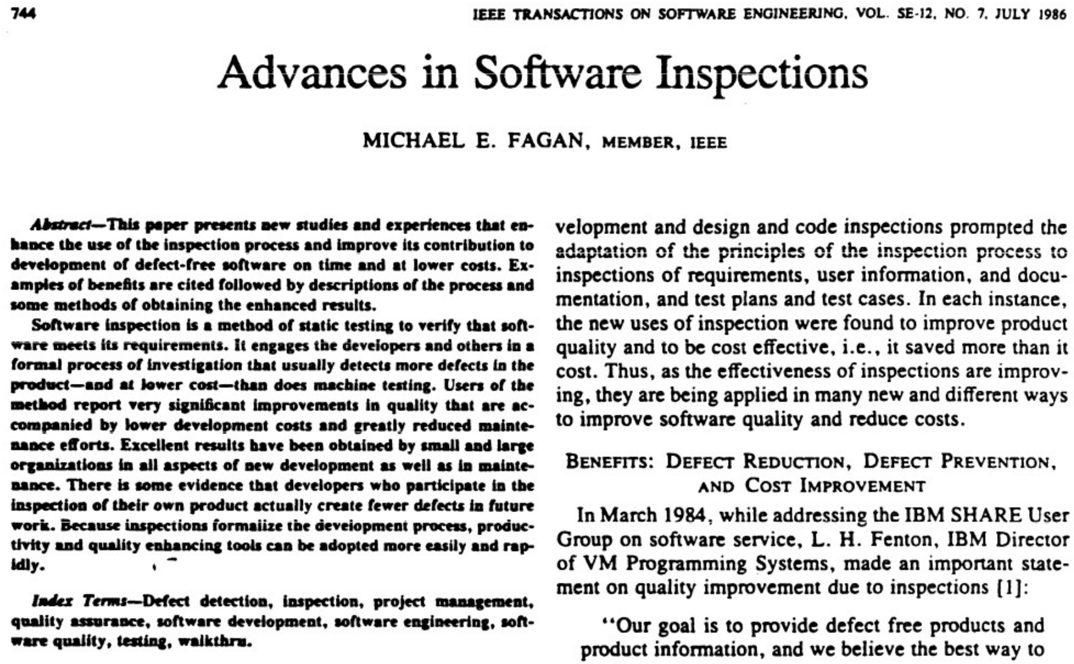
O que é **Qualidade de Software** ?



Design and code inspections to reduce errors in program development

by M. E. Fagan

Successful management of any process requires planning, measurement, and control. In programming development, these requirements translate into defining the programming process in terms of a series of operations, each operation having its own exit criteria. Next there must be some means of measuring completeness of the product at any point of its development by inspections or testing. And finally, the measured data must be used for controlling the process. This approach is not only conceptually interesting, but has been applied successfully in several programming projects embracing systems and applications programming, both large and small. It has not been found to "get in the way" of programming, but has instead enabled higher predictability than other means, and the use of inspections has improved productivity and product quality. The purpose of this paper is to explain the planning, measurement, and control functions as they are affected by inspections in programming terms.



Advances in Software Inspections

MICHAEL E. FAGAN, MEMBER, IEEE

Abstract—This paper presents new studies and experiences that enhance the use of the inspection process and improve its contribution to development of defect-free software on time and at lower costs. Examples of benefits are cited followed by descriptions of the process and some methods of obtaining the enhanced results.

Software inspection is a method of static testing to verify that software meets its requirements. It engages the developers and others in a formal process of investigation that usually detects more defects in the product—and at lower cost—than does machine testing. Users of the method report very significant improvements in quality that are accompanied by lower development costs and greatly reduced maintenance efforts. Excellent results have been obtained by small and large organizations in all aspects of new development as well as in maintenance. There is some evidence that developers who participate in the inspection of their own product actually create fewer defects in future work. Because inspections formaiize the deveiopment process, productivity and quality enhancing tools can be adopted more easily and rapidly.

Index Terms—Defect detection, inspection, project management, quality assurance, software development, software engineering, software quality, testing, walkthru.

velopment and design and code inspections prompted the adaptation of the principles of the inspection process to inspections of requirements, user information, and documentation, and test plans and test cases. In each instance, the new uses of inspection were found to improve product quality and to be cost effective, i.e., it saved more than it cost. Thus, as the effectiveness of inspections are improving, they are being applied in many new and different ways to improve software quality and reduce costs.

Benefits: Defect Reduction, Defect Prevention, and Cost Improvement

In March 1984, while addressing the IBM SHARE User Group on software service, L. H. Fenton, IBM Director of VM Programming Systems, made an important statement on quality improvement due to inspections [1]:

> "Our goal is to provide defect free products and product information, and we believe the best way to

Qual é a **#1 avitivdade que uma empresa deve realizar para garantir a Qualidade de Sofware ?**

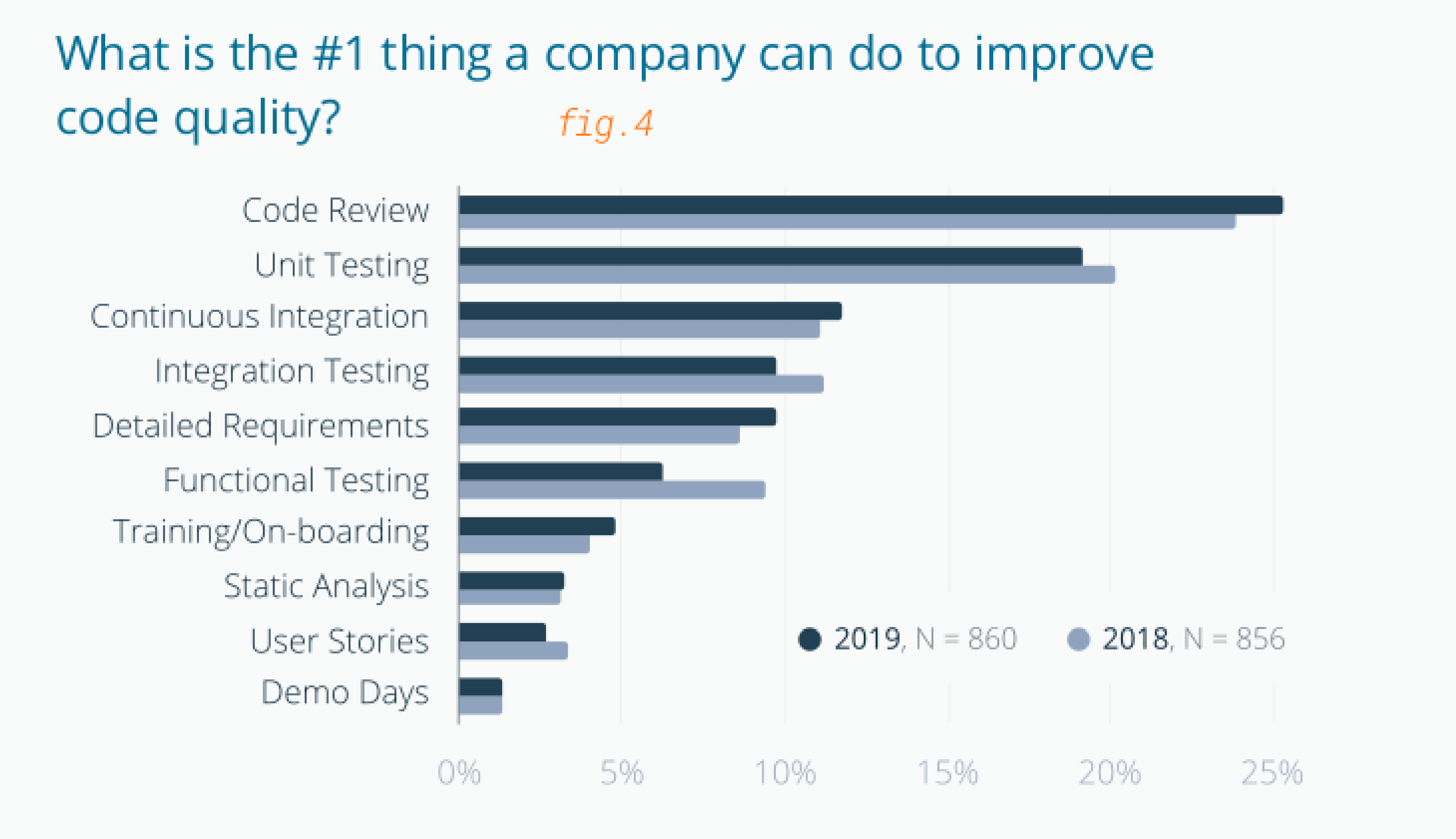


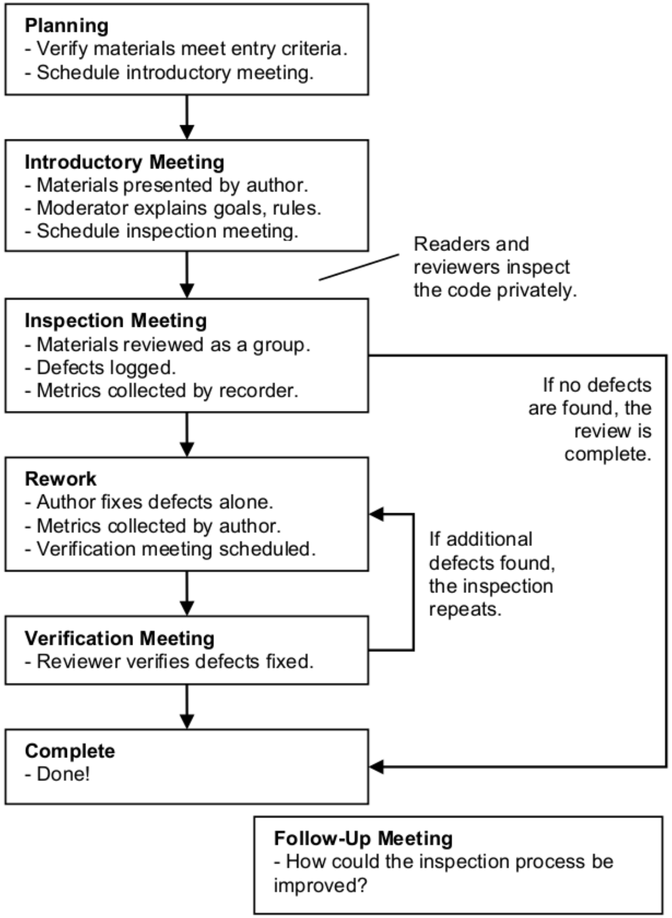
fig.4

SMARTBEAR (2019). Disponível em <https://static1.smartbear.co/smartbearbrand/media/pdf/the-2019-state-of-code-review.pdf>

Tipos de **Code Review**

- **Inspeções Formais**
  - Fagan

A Typical Formal Inspection Process



Planning
- Verify materials meet entry criteria.
- Schedule introductory meeting.

Introductory Meeting
- Materials presented by author.
- Moderator explains goals, rules.
- Schedule inspection meeting.

Readers and reviewers inspect the code privately.

Inspection Meeting
- Materials reviewed as a group.
- Defects logged.
- Metrics collected by recorder.

If no defects are found, the review is complete.

Rework
- Author fixes defects alone.
- Metrics collected by author.
- Verification meeting scheduled.

If additional defects found, the inspection repeats.

Verification Meeting
- Reviewer verifies defects fixed.

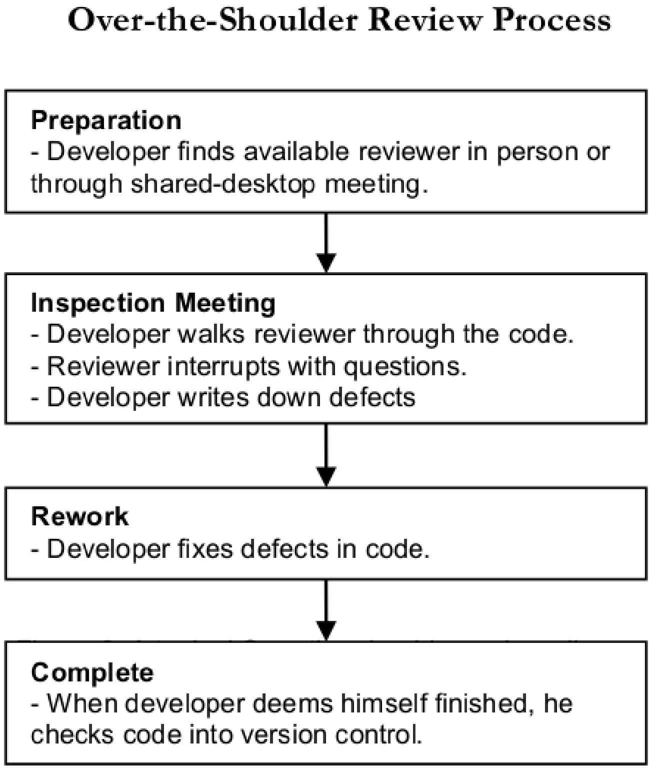
Complete
- Done!

Follow-Up Meeting
- How could the inspection process be improved?

SMARTBEAR (2019)

Tipos de **Code Review**

- **Over-the-shoulder reviews**



Over-the-Shoulder Review Process

Preparation
- Developer finds available reviewer in person or through shared-desktop meeting.

Inspection Meeting
- Developer walks reviewer through the code.
- Reviewer interrupts with questions.
- Developer writes down defects

Rework
- Developer fixes defects in code.

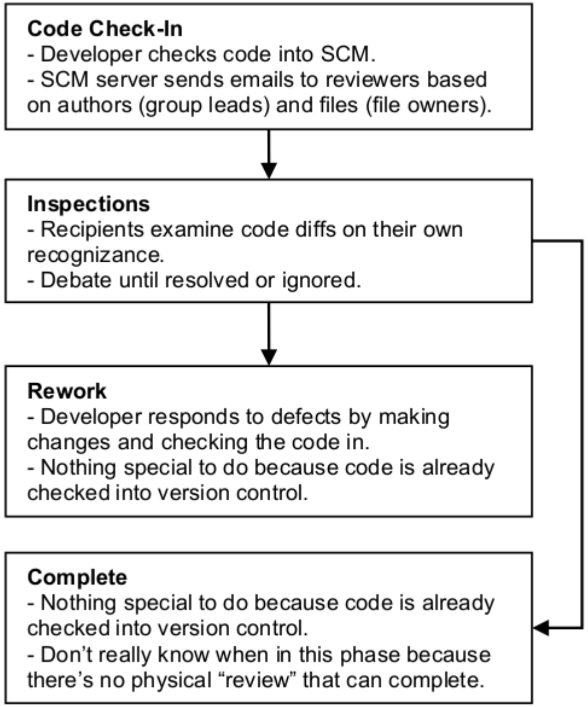
Complete
- When developer deems himself finished, he checks code into version control.

SMARTBEAR (2019)

Tipos de **Code Review**

- **E-mail pass-around reviews**

E-Mail Pass-Around Process: Post Check-In Review



Code Check-In
- Developer checks code into SCM.
- SCM server sends emails to reviewers based on authors (group leads) and files (file owners).

Inspections
- Recipients examine code diffs on their own recognizance.
- Debate until resolved or ignored.

Rework
- Developer responds to defects by making changes and checking the code in.
- Nothing special to do because code is already checked into version control.

Complete
- Nothing special to do because code is already checked into version control.
- Don't really know when in this phase because there's no physical "review" that can complete.

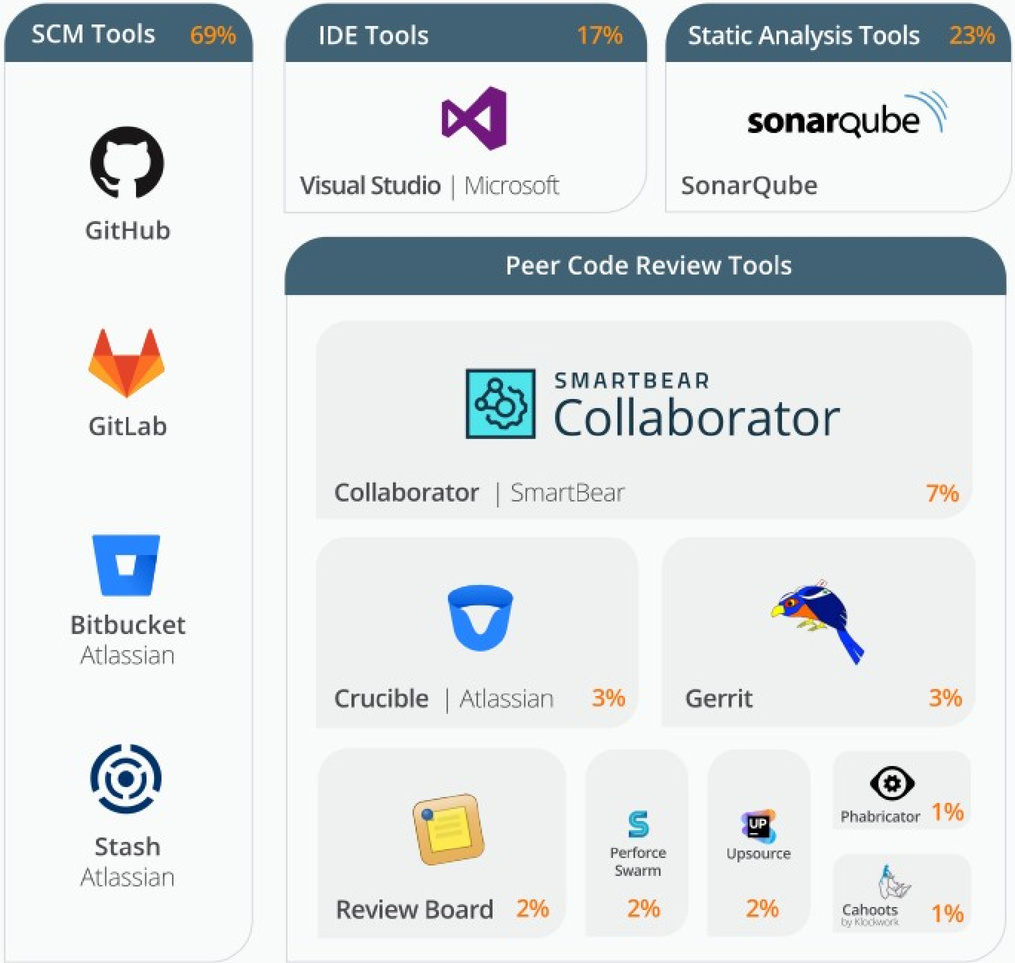
SMARTBEAR (2019)

Tipos de **Code Review**

- **Tool-Assisted reviews**

fig. 14

N = 905



SCM Tools 69%

GitHub

GitLab

Bitbucket
Atlassian

Stash
Atlassian

IDE Tools 17%

Visual Studio | Microsoft

Static Analysis Tools 23%

SonarQube

Peer Code Review Tools

Collaborator | SmartBear 7%

Crucible | Atlassian 3%

Gerrit 3%

Review Board 2%

Perforce Swarm 2%

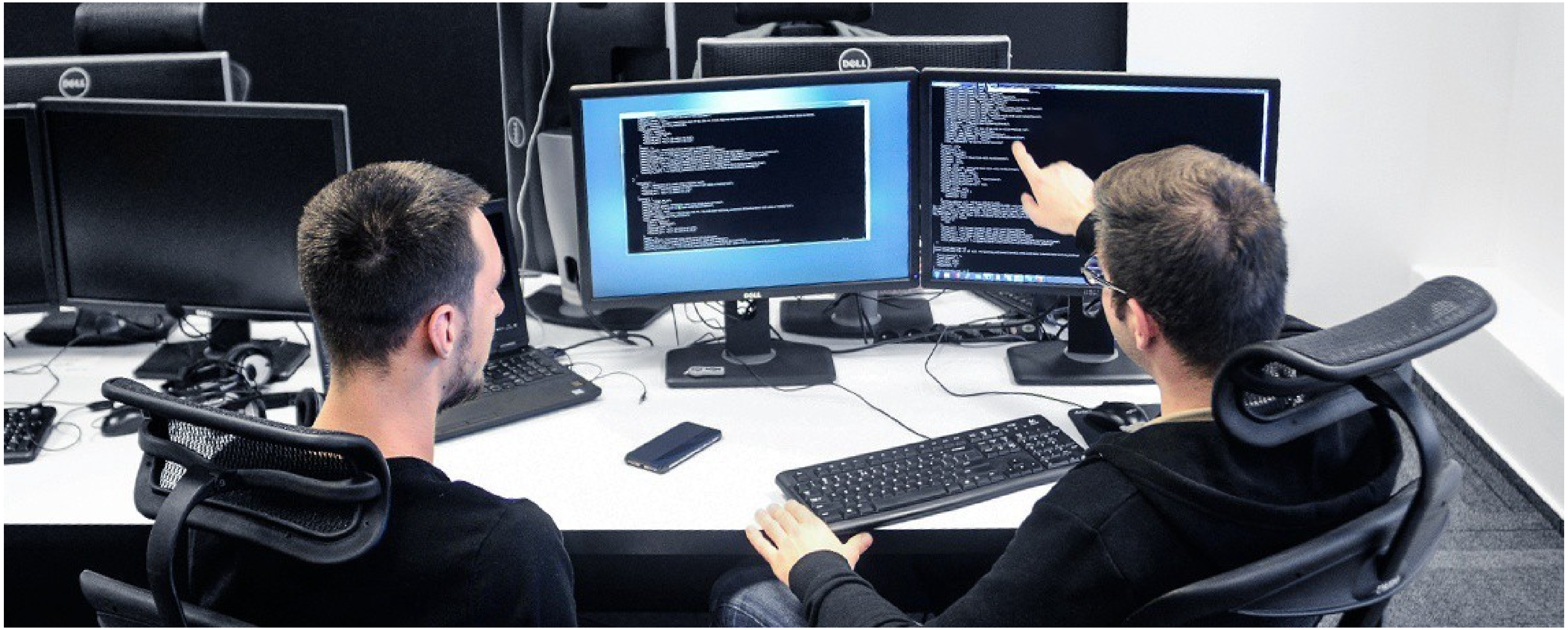
Upsource 2%

Phabricator 1%

Cahoots by Klocwork 1%

Tipos de **Code Review**

- **Pair-Programming**



Code Review com Checklist

deteca até **30%**

mais defeitos

que outras atividades estáticas

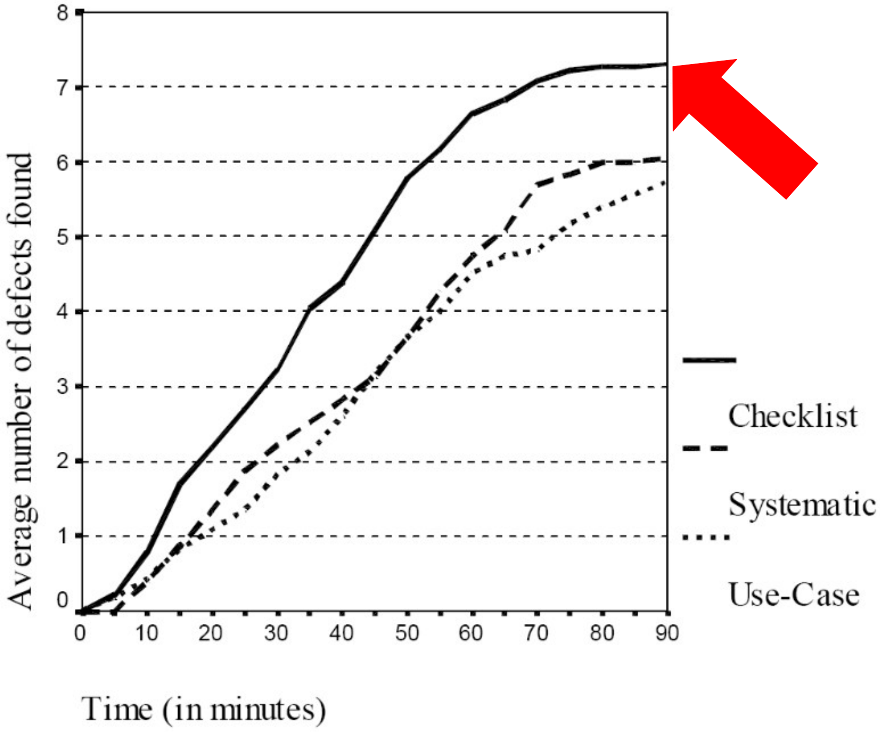
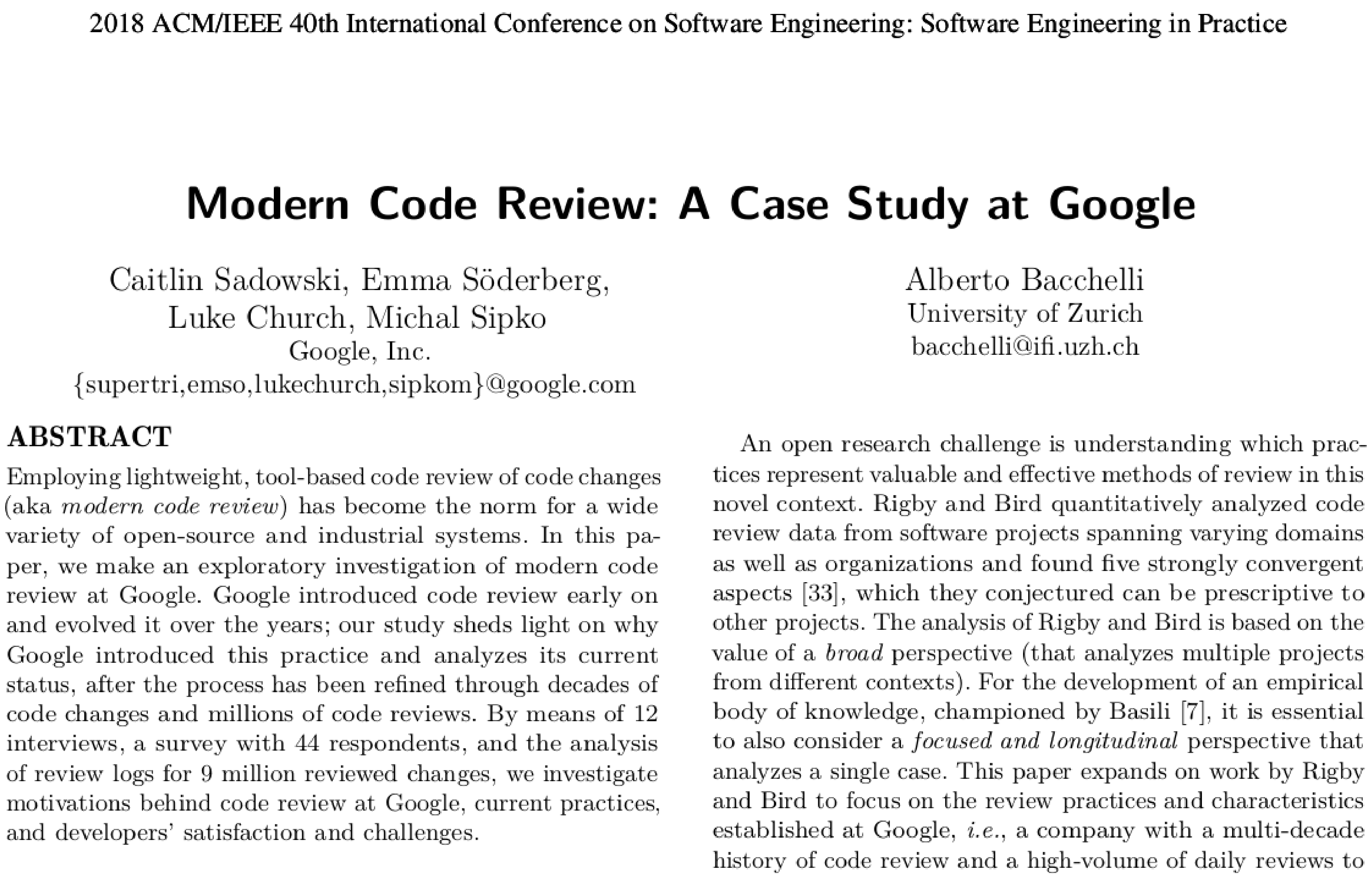


Figure 7: Elapsed time versus cumulative number of defects found for each of the three types of review.

The defect rate is constant until about 60 minutes into the inspection at which point it levels off with no defects found at all after 90 minutes.

Dunsmore, A., Roper, M., Wood, M. Object-Oriented Inspection in the Face of Delocalisation, appeared in Proceedings of the 22nd International Conference on Software Engineering (ICSE) 2000, pp. 467-476, June 2000



2018 ACM/IEEE 40th International Conference on Software Engineering: Software Engineering in Practice

Modern Code Review: A Case Study at Google

Caitlin Sadowski, Emma Söderberg,
Luke Church, Michal Sipko
Google, Inc.
{supertri,emso,lukechurch,sipkom}@google.com

Alberto Bacchelli
University of Zurich
bacchelli@ifi.uzh.ch

ABSTRACT

Employing lightweight, tool-based code review of code changes (aka *modern code review*) has become the norm for a wide variety of open-source and industrial systems. In this paper, we make an exploratory investigation of modern code review at Google. Google introduced code review early on and evolved it over the years; our study sheds light on why Google introduced this practice and analyzes its current status, after the process has been refined through decades of code changes and millions of code reviews. By means of 12 interviews, a survey with 44 respondents, and the analysis of review logs for 9 million reviewed changes, we investigate motivations behind code review at Google, current practices, and developers' satisfaction and challenges.

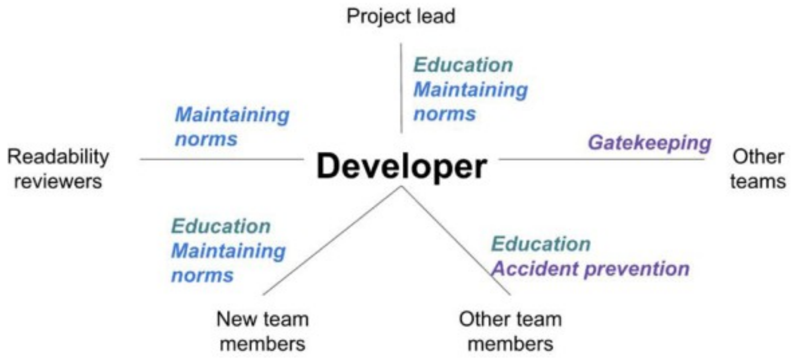
An open research challenge is understanding which practices represent valuable and effective methods of review in this novel context. Rigby and Bird quantitatively analyzed code review data from software projects spanning varying domains as well as organizations and found five strongly convergent aspects [33], which they conjectured can be prescriptive to other projects. The analysis of Rigby and Bird is based on the value of a *broad* perspective (that analyzes multiple projects from different contexts). For the development of an empirical body of knowledge, championed by Basili [7], it is essential to also consider a *focused and longitudinal* perspective that analyzes a single case. This paper expands on work by Rigby and Bird to focus on the review practices and characteristics established at Google, *i.e.*, a company with a multi-decade history of code review and a high-volume of daily reviews to

Modern Code Review

> “Modern code review is
>
> **(1)** informal (in contrast to Fagan-style)
>
> **(2)** tool-based,
>
> **(3)** asynchronous and;
>
> **(4)** focused on reviewing code changes.

Premissas segundo o Google
Code Review

- Detectar defeitos é bem vindo, **mas não é só isso**
  - **É também sobre comapartilhar conhecimento**
  - **Histórico de mudanças**
  - **Retrospectiva**



Premissas segundo o Google
Code Review

- **Processo de Code Review**
  - **Criação do Código**
  - **Pre-visualização do Código – ferramentas locais**
    - git diff
  - **Comentários – Pull Request via Git**
  - **Atribuir um Feedback sobre o código**
  - **Aprovação do Código**

Premissas segundo o Google **Code Review**

- **Tamanho do Código**
  - Código de Tamanho pequeno
  - 35 % das revisões alteram um único arquivo
  - 90% das revisões em menos de 10 arquivos
  - 10% das mudanças alteram uma linha de código
  - 24 linhas de código são alteradas na média

Premissas segundo o Google
Code Review

- **Quanidade de Revisores**
  - **25% das mudanças** requerem ao menos **1 revisor**
  - **99% das mudanças** tem ao menos **>=5 revisores**
  - **Alterações pesadas >=2 revisores**

Exemplo