Secure Code Reviews

Dr. Michaela Greiler

Increasing your Code Review Superpower

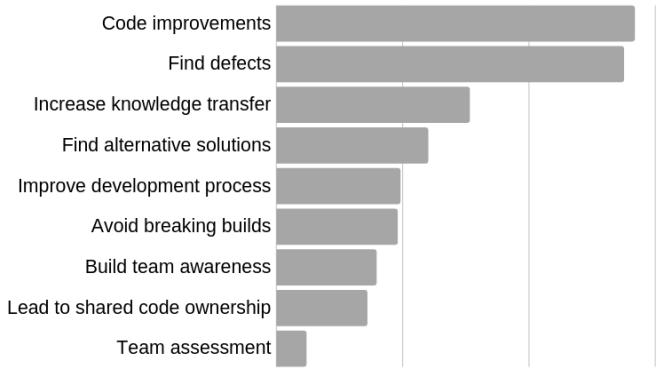
More information at michaelagreiler.com

hi@michaelagreiler.com

1

Many companies invest in Code Reviews

Motivations for Code Reviews @Microsoft



Microsoft

Code Reviews Do Find Bugs

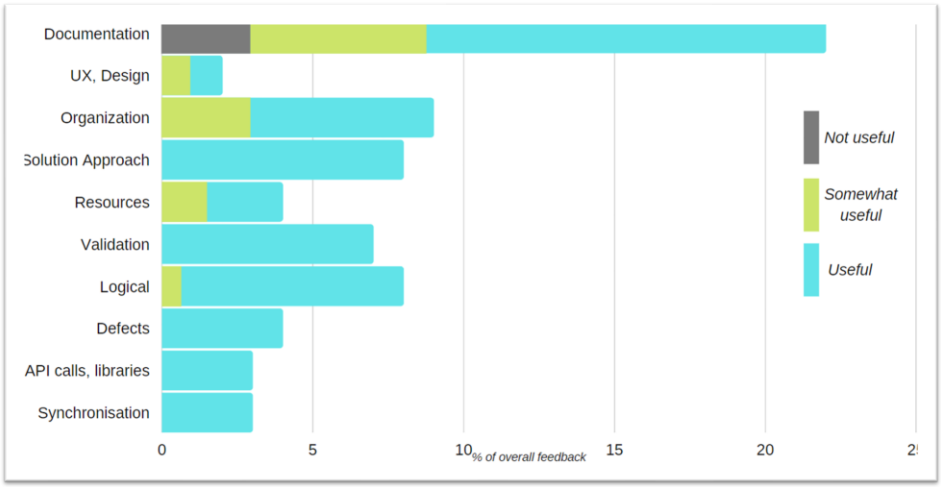
Code reviews correlate with a reduction of defects.

Unreviewed code is 2X more likely to introduce defects than reviewed code.

At Google, 80% of code reviews lead to code improvements.

Sources:
Fagan, Kemerer and Paulk, Tanaka et al., Bavota and Russo, Thongtanunam et al., Bacchelli and Sadowski.

Not all code review feedback is equal!



Source: Characteristics of useful code reviews: an empirical study at Microsoft, Bosu, Greiler, Bird

5

Code Review Feedback

Best:

Functional defects,
missing corner cases or validation,
Api usage, best practices

OK:

Documentation,
coding style & conventions,
spelling mistakes

No-go:

Alternatives without benefits,
existing tech debt and refactoring,
planning and future work

What to focus on during code reviews

- **Is the Code Correct?**

  Does the code do what it's supposed to? Does it handle edge cases? Is it adequately tested to make sure that it stays correct even when other engineers modify it? Is it performant enough for this use case?

- **Is the Code Secure**?

  Does the code have vulnerabilities? Is the data stored safely? Is personal identification information (PII) handled correctly? Could the code be used to induce a DOS? Is input validation comprehensive enough?

- **Is the Code Readable**?

  Is the code easy to read and comprehend? Does it make clear what the business requirements are (code is written to be read by a human, not by a computer)? Are tests concise enough? Are variables, functions and classes named appropriately? Do the domain models cleanly map the real world to reduce cognitive load? Does it use consistent coding convention?

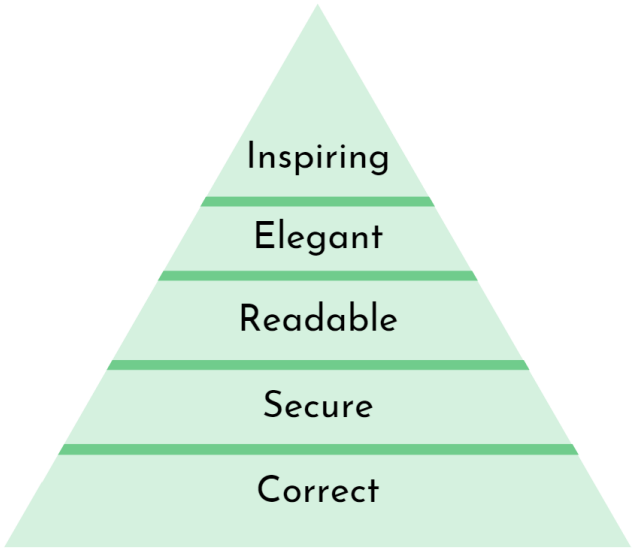
- **Is the Code Elegant**?

  Does the code leverage well-known patterns? Does it achieve what it needs to do without sacrificing simplicity and conciseness? Would you be excited to work in this code? Would you be proud of this code?

- **Is the Code Inspiring?**

  Does the code leave the codebase better than what it was? Does it inspire other engineers to improve their code as well? Is it cleaning up unused code, improving documentation, introducing better patterns through small-scale refactoring?

7



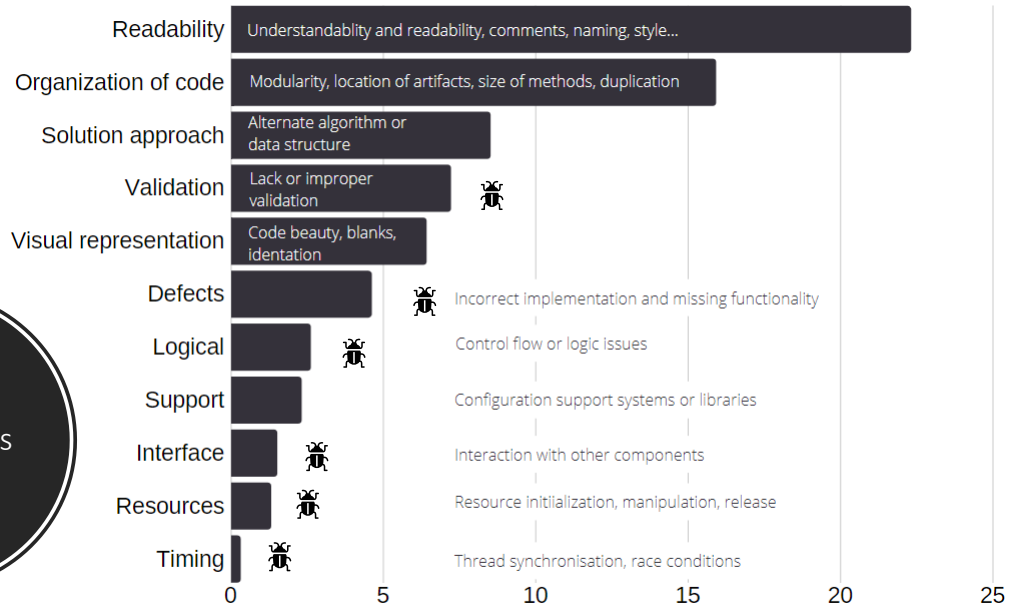
https://www.reddit.com/r/programming/comments/2wau2x/maslows_pyramid_of_code_review/

Code Reviews Are Not Only About Bugs

15% of the comments are about defects

8

What are code reviews about?

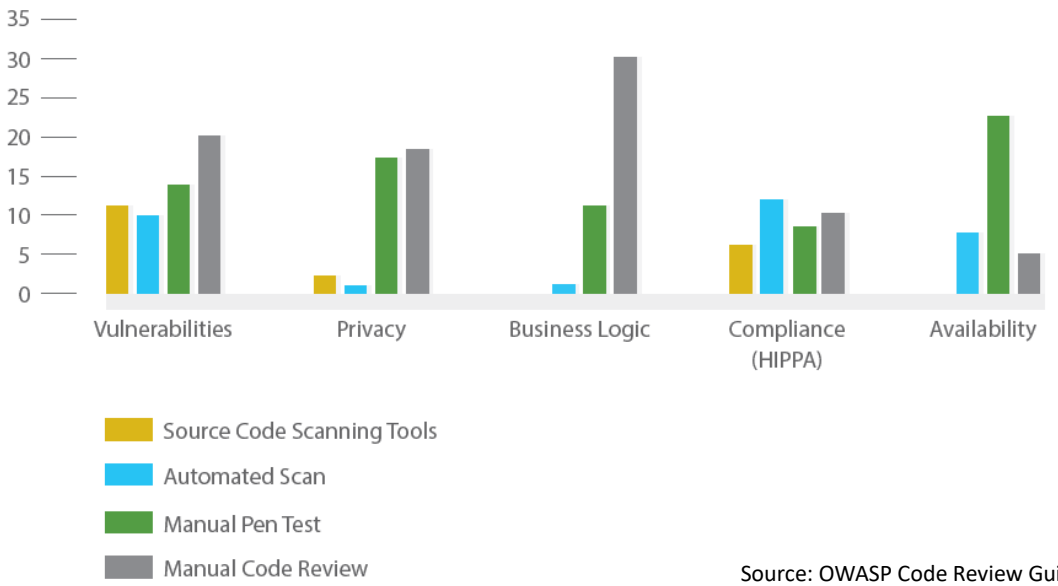


Sources:

- Bosu, Greiler, Bird: Characteristics of Useful Code Reviews: An Empirical Study at Microsoft 2015
- Mäntylä and Lassenius. What Types of Defects Are Really Discovered in Code Reviews? 2009

9

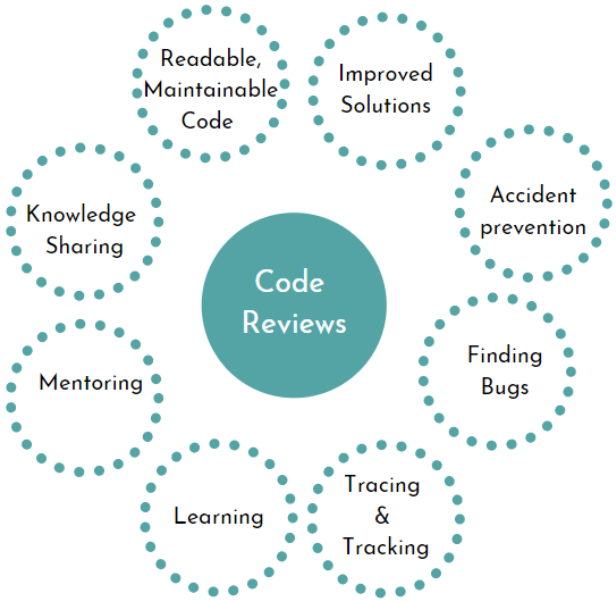
Figure 1: Survey relating detection methods to general vulnerability types



Source: OWASP Code Review Guide 2.0

10

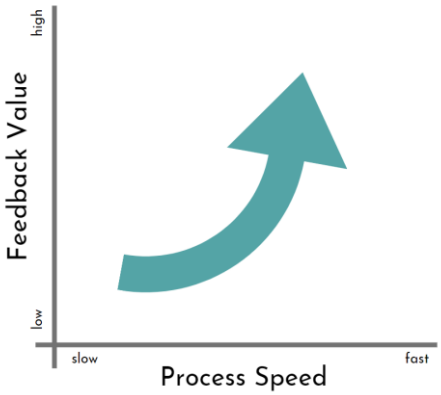
Code Reviews are so much more

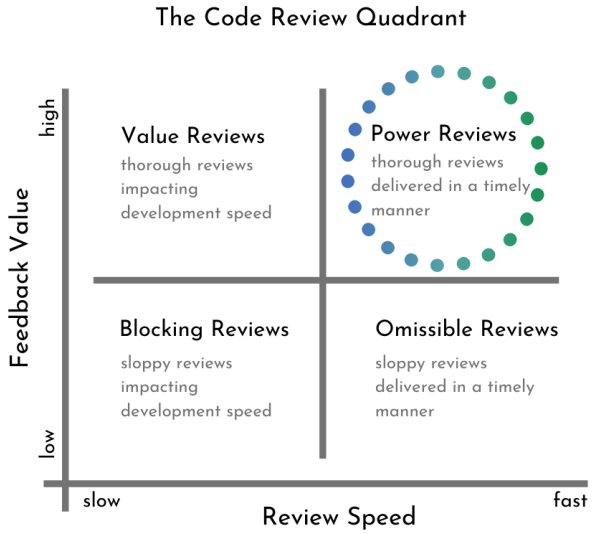


Main Pain Points

Slow Turn-Around Time

Low Review Quality



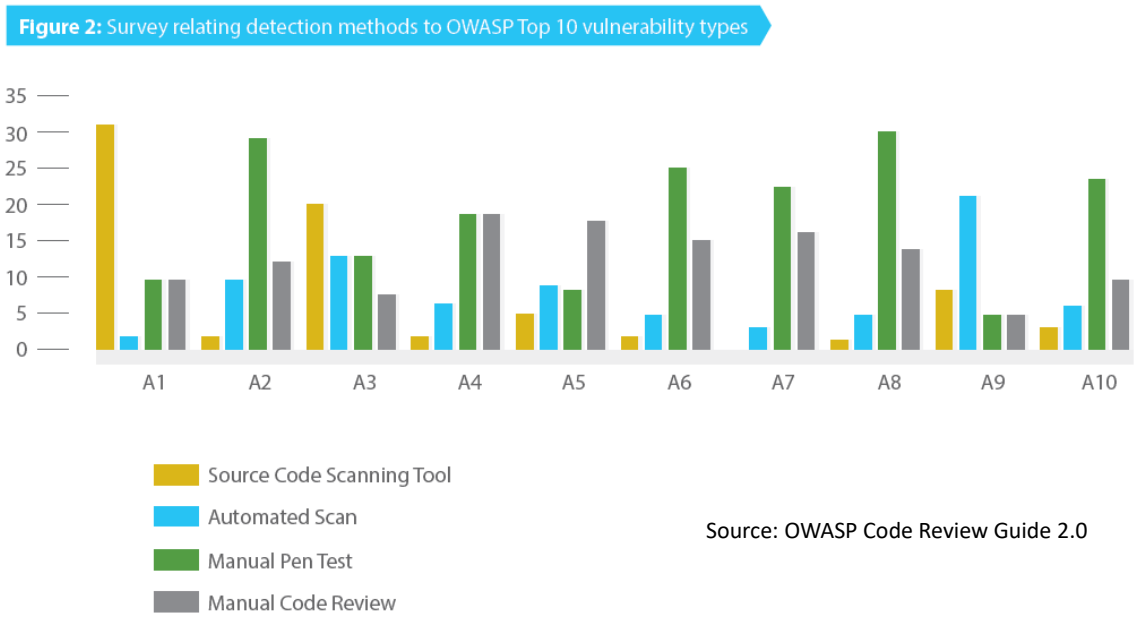


How do we make sure we find important issues?

15

Know common security vulnerabilities

- OWASP Top 10 – Web app centric: https://owasp.org/www-project-top-ten/
  - Injection
  - Broken Authentication
  - Sensitive Data Exposure
  - XML External Entities (XXE)
  - Broken Access Control
  - Security Misconfiguration
  - Cross-Site Scripting (XSS)
  - Insecure Deserialization
  - Using Components with Known Vulnerabilities
  - Insufficient Logging & Monitoring
  - Code review guide: https://owasp.org/www-pdf-archive/OWASP_Code_Review_Guide_v2.pdf
- Common Weaknesses by MITRE: http://cwe.mitre.org/data/
  - 40 categories comprising 418 weaknesses
  - Top 25 Most Dangerous Software Weaknesses



Source: OWASP Code Review Guide 2.0

Implementation

- Does this code change do what it is supposed to do?
- Can this solution be simplified?
- Does this change add unwanted compile-time or run-time dependencies?
- Was a framework, API, library, service used that should not be used?
- Was a framework, API, library, service not used that could improve the solution?
- Is the code at the right abstraction level?
- Is the code modular enough?
- Would you have solved the problem in a different way that is substantially better in terms of the code's maintainability, readability, performance, security?
- Does similar functionality already exist in the codebase? If so, why isn't this functionality reused?
- Are there any best practices, design patterns or language-specific patterns that could substantially improve this code?
- Does this code follow Object-Oriented Analysis and Design Principles, like the Single Responsibility Principle, Open-close Principle, Liskov Substitution Principle, Interface Segregation, Dependency Injection?

Logic Errors and Bugs

- Can you think of any use case in which the code does not behave as intended?

Dependencies

- If this change requires updates outside of the code, like updating the documentation, configuration, readme files, was this done?
- Might this change have any ramifications for other parts of the system, backward compatibility?

Security and Data Privacy

- Does this code open the software for security vulnerabilities?
- Are authorization and authentication handled in the right way?
- Is sensitive data like user data, credit card information securely handled an stored?
- Is the right encryption used?
- Does this code change reveal so secret information like keys, pa or usernames?
- If code deals with user inpu address security vulnerab cross-site scripting, SQ it do input sanitizati data retrieved fro libraries check

Readabi

- Wa

Use a Code Review Checklist

https://github.com/mgreiler/secure-code-review-checklist

Security and Data Privacy

- What security vulnerabilities is this code susceptible to?
- Are authorization and authentication handled in the right way?
- Is (user) input validated, sanitized, and escaped to prevent security attacks such as cross-site scripting, SQL injection?
- Is sensitive data like user data, credit card information securely handled and stored?
- Does this code change reveal some secret information like keys, passwords, or usernames?
- Is data retrieved from external APIs or libraries checked accordingly?
- Is the right encryption used?

Web Security Principles

Authentication	Confirm something is authentic. Example: confirming the identity of a user.
Authorization	Specify access rights to resources. Example: only Joe can view Joe's account balance.
Confidentiality	Prevent the disclosure of information to unauthorized individuals or systems. Example: message encryption.
Data / Message Integrity	Data cannot be modified or corrupted without detection.
Availability	Web sites need to be available and fast. Example: many websites can boast 99.99% uptime.
Accountability	When a person or system accesses or changes data their actions should be traceable. Example: logging
Non-repudiation	The ability to prove that a transaction took place. Example: electronic receipts.

32

Finding vulnerabilities is hard

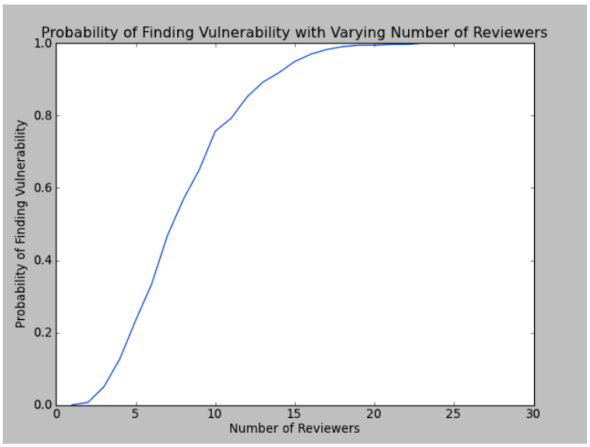
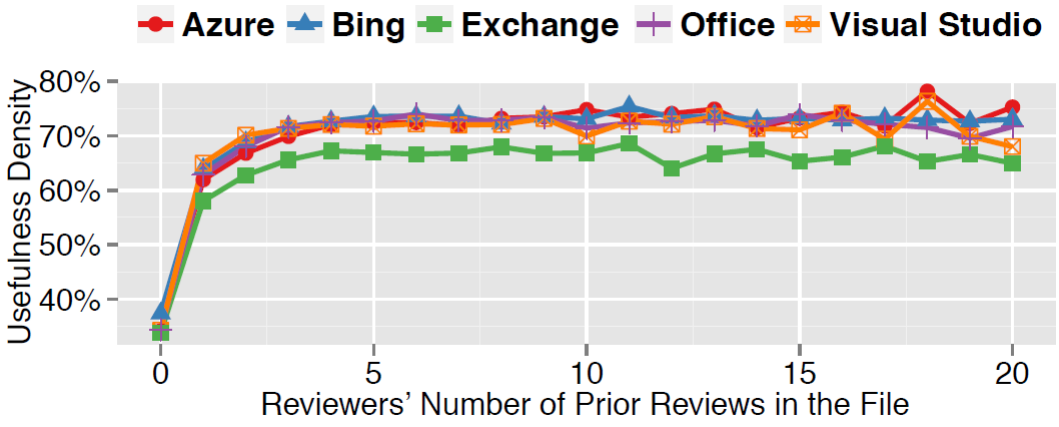


Fig. 4. A graph showing the probability of finding all vulnerabilities depending on the number of

Source: An Empirical Study on the Effectiveness of Security Code Review, Edmundson et. al

33


Azure
Bing
Exchange
Office
Visual Studio
Usefulness Density
80%
70%
60%
50%
40%
0
5
10
15
20
Reviewers' Number of Prior Reviews in the File


Understanding the Context

Focus on WHAT and WHY, not HOW!

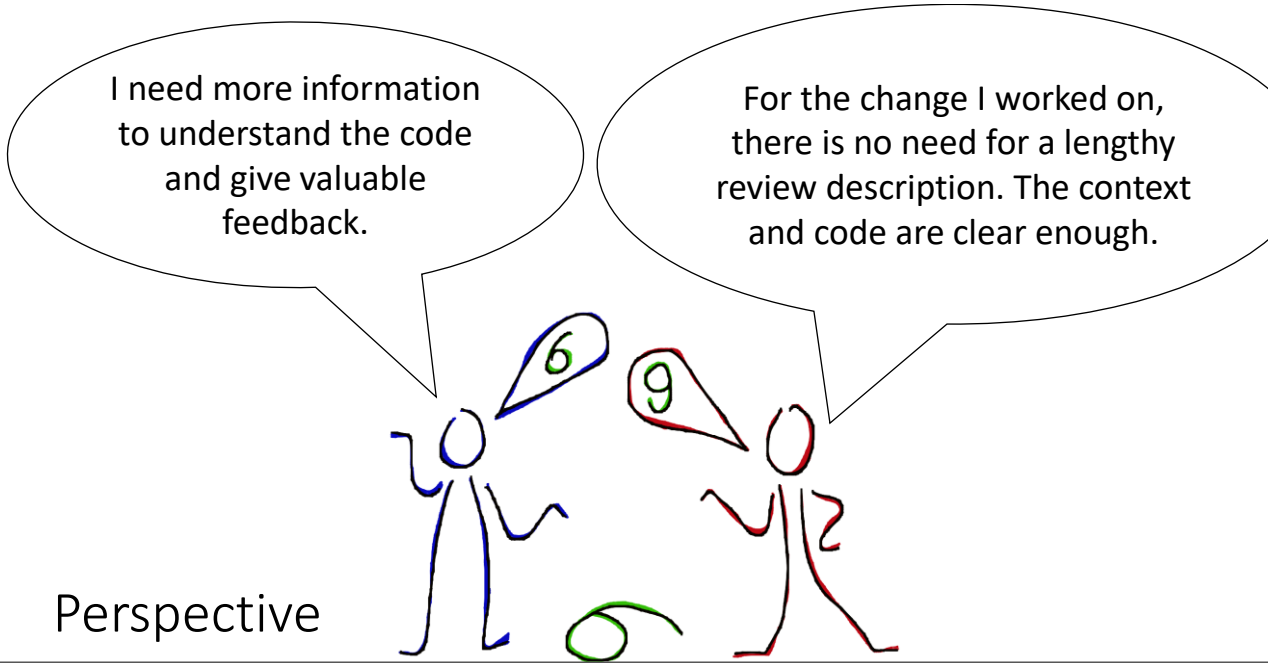
- What does this change accomplish?
- Why was this change necessary?
- Why did you come up with this solution?
- Have you considered alternative solutions?
- Why did you decide against them?

36

Give Context and Lead The Review

- What is the entry point of your solution?
- Which order of files makes most sense for the reviewer?
- Would a gif, video or screenshot help understand the change?
- Add comments to get a reviewer's attention
- Add comments to ask for specific feedback

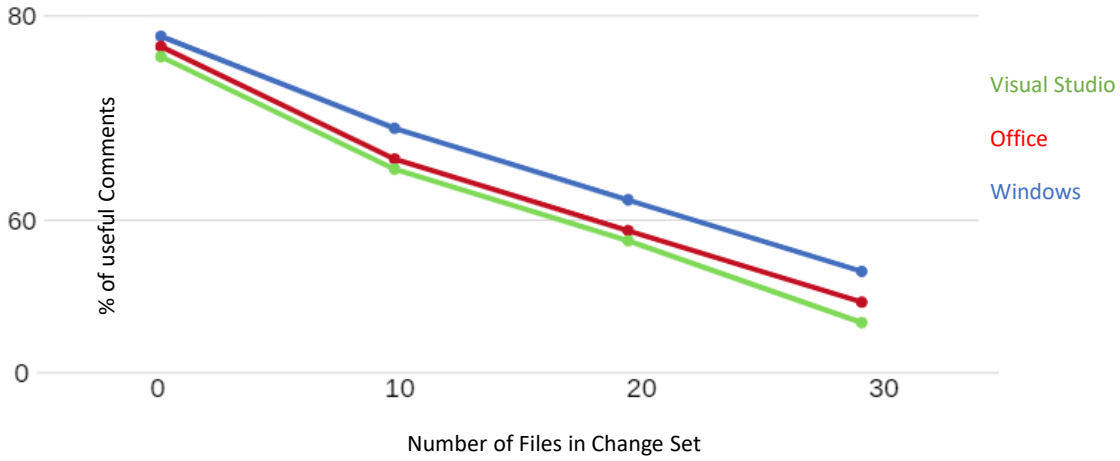
37



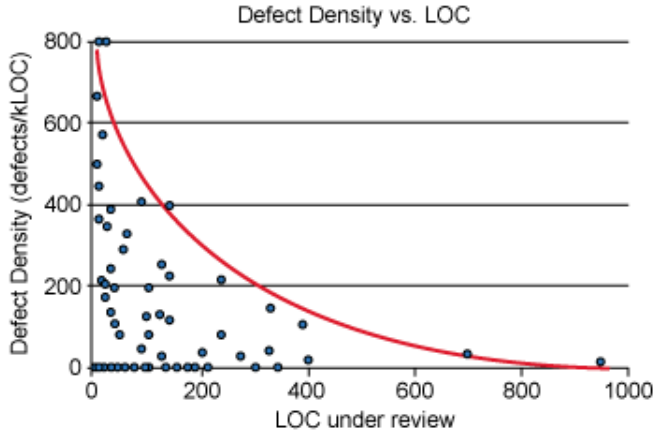
Perspective

Change Size

Code Review Size – Feedback Quality



Code Review Size – Feedback Quality



200-400 LOC are the maximum a developer can effectively process

Source: Best Kept Secrete of Code Reviews a study of Cisco's code review

Large Pull Requests

I Am Devloper
@iamdevloper

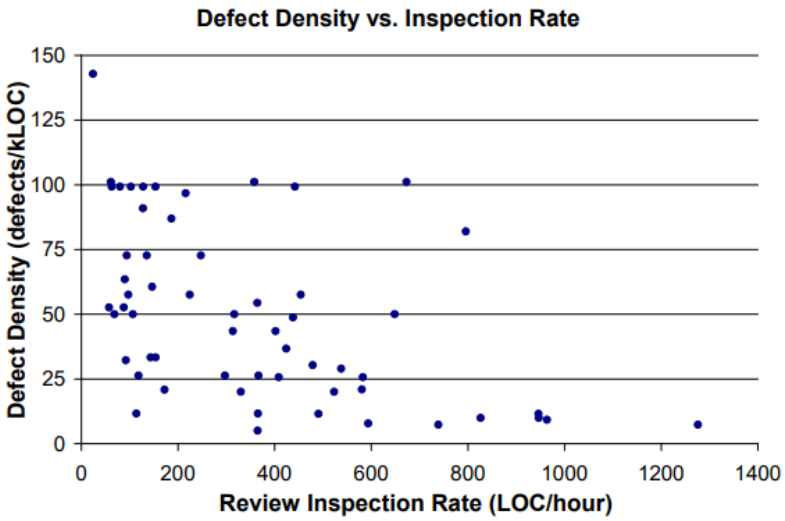
10 lines of code = 10 issues.

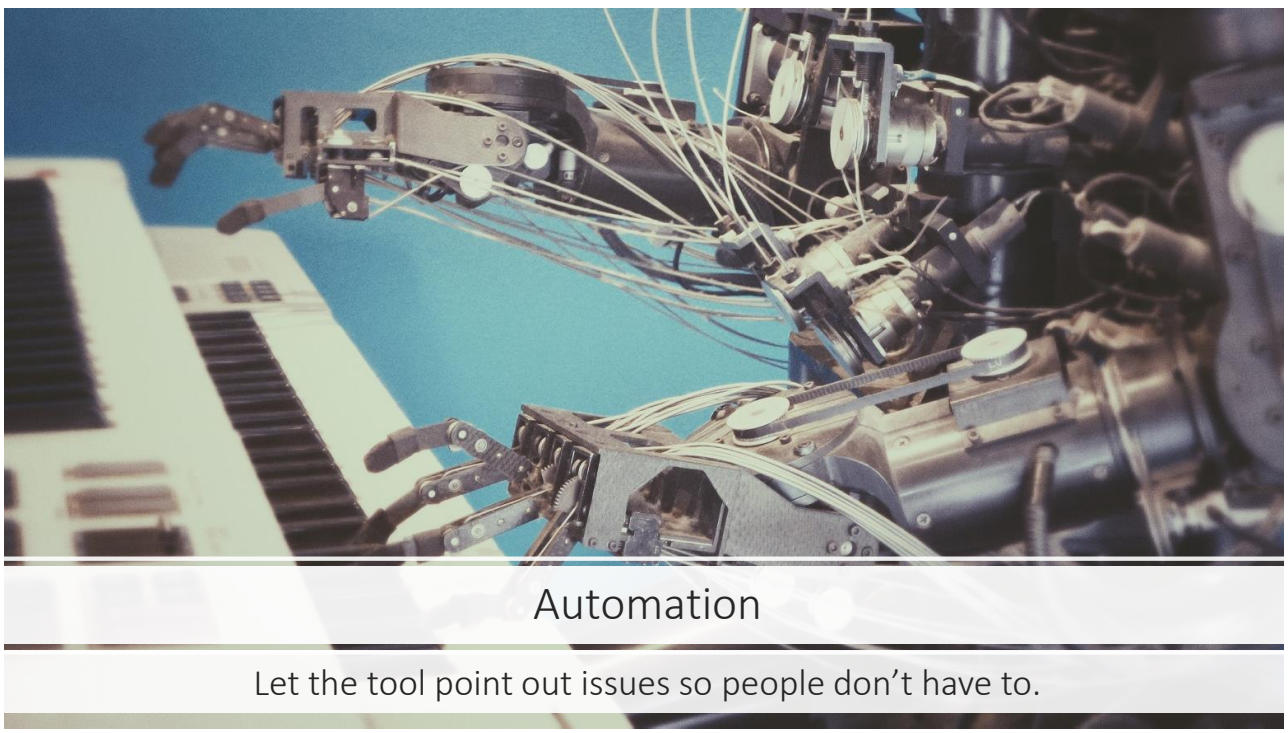
500 lines of code = "looks fine."

Code reviews.

42

43





Automation

Let the tool point out issues so people don't have to.

Integrating Linting on the DEVOPs cycle

Linter and other tools should be part of an automated loop

Linting can take time. Design tool chain to reduce interruptions and waiting times.

But make sure problems are reported before merge.

46

Problems reported after merge don’t get fixed

47

Automatic Scanning

Strength

- Can runs continuously with CI
- Finds buffer overflows, SQL Injection Flaws
- Helps pinpoint developers to problematic files

Weaknesses

- Isn't good in finding authentication, access control, or cryptography problems
- Reports many false positives
- Can only scan code (i.e., config can be problematic if not present in code)
- Code must be compile/runnable

How to make sure we find important issues?


Use a code review checklist


Learn about security issues


Include people with the right expertise/experience on the review


Set enough time aside for a review


Review small, incremental changes


Learn, learn, learn

Let's do a code review

50

Focus of a secure code review

- Data Validation
- Error Handling
- Logging
- Authentication
- Session Management
- Authorization
- Cryptography

51

Data & Input Validation

- All data from users needs to be considered untrusted.
- Best practices:
  - Exact match validator
  - „Known good" approach (allowed list)
  - "Known bad" approach (block list).
- Input data: not only user data but also HTTP headers, input fields, hidden fields, drop down lists, and other web components
- Check: type, length, characters.
- Do contextual escaping, instead of replacment
- Always validate on the server side (again)
- Use parameterized queries

Improper input validation can lead to

- Cross-site scripting (XSS) (CWE-79) attack
- SQL injection (CWE-89).
- Carriage Return Line Feed (CRLF) Injection (CWE-93)
- Argument Injection (CWE-88)
- Command Injection (CWE-77)

- Learn more: http://cwe.mitre.org/data/definitions/20.html

SQL Injection

```
$query = "SELECT * FROM users WHERE name = '{$name}'"
```

Input: Michaela; DROP TABLE users;

54

Authentication

- Can admin accounts log-in via the web?
- Are failure messages for invalid usernames or passwords leak information?
- Are invalid passwords logged (which can leak sensitive pwd & username combis)?
- Are the pwd requirements (lengths/complexity) appropriated?
- Are invalid login attempts correctly handled with lockouts, and rate limit?
- Does the "forgot pwd" routine leak information, is vulnerable to spamming, or is the pwd send in plain text via email?
- How and where are pwd and usernames stored, and are appropriate mechanisms such as hashing, salts, encryption in place?
- …

- *More info: https://cheatsheetseries.owasp.org/cheatsheets/Authentication_Cheat_Sheet.html*

57

Experience the problems

58

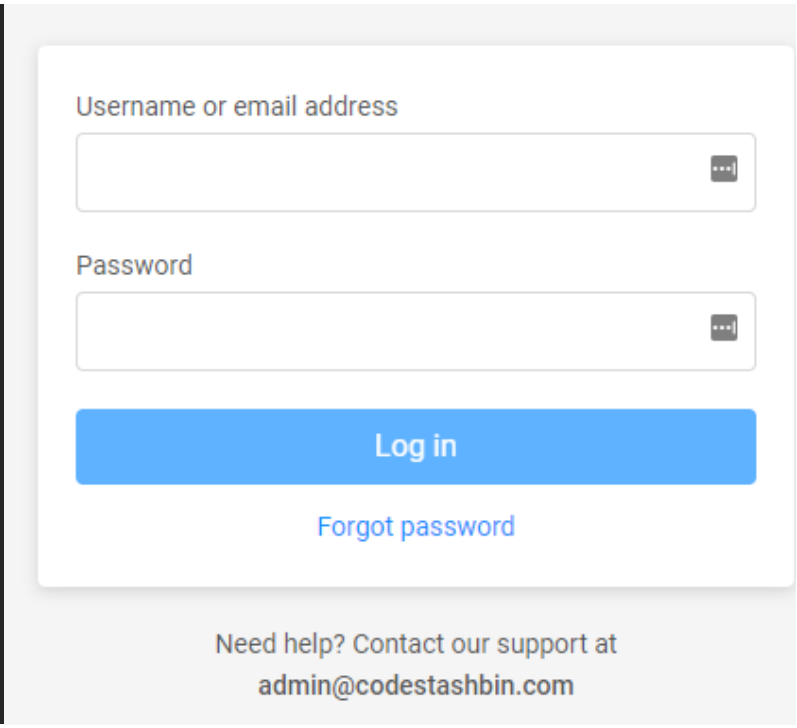
Password reset routine

- Notice that the reset password email is sent to the email address supplied in the request, but not the one retrieved from the database.

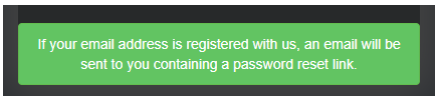
```
app.post('/api/resetPassword', function(req, res){
  const email = req.body.email;
  if(!email) {
    res.status(400).send('Missing email in request body');
  }

  accountRepository.findUserByEmail(email, (user) => {
    if(user != null) {
      passwordGenerator.generateTemporaryPassword((tempPassword) => {
        accountRepository.resetPassword(email, tempPassword, () => {

          messenger.sendPasswordResetEmail(email, tempPassword, res);

          res.status(204).send();
        });
      });
    } else
      res.status(204).send();
  });
});
```

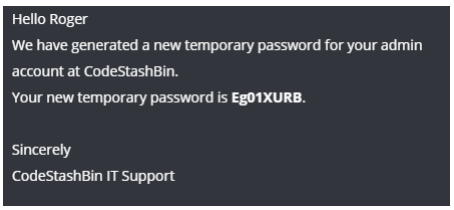
59

Give away info for exploitation



60





61

Learn: Language Agnostic and Specific Security

- Code Review Guide: https://owasp.org/www-pdf-archive/OWASP_Code_Review_Guide_v2.pdf

- Ruby on Rails: https://rails-sqli.org/
- Rails Security Guide: https://guides.rubyonrails.org/security.html

- Great resources:
- Secure Coding Practices Checklist: https://owasp.org/www-pdf-archive/OWASP_SCP_Quick_Reference_Guide_v2.pdf
- Cheat Sheets: https://cheatsheetseries.owasp.org/cheatsheets/SQL_Injection_Prevention_Cheat_Sheet.html
- Input Validation: https://owasp-top-10-proactive-controls-2018.readthedocs.io/en/latest/c5-validate-all-inputs.html

62

Questions & Discussion

@mgreiler

michaelagreiler.com

hi@michaelagreiler.com

Get Dr. Greiler's
Code Review E-Book
including Code Review Checklists

bit.ly/CRE-Book

63