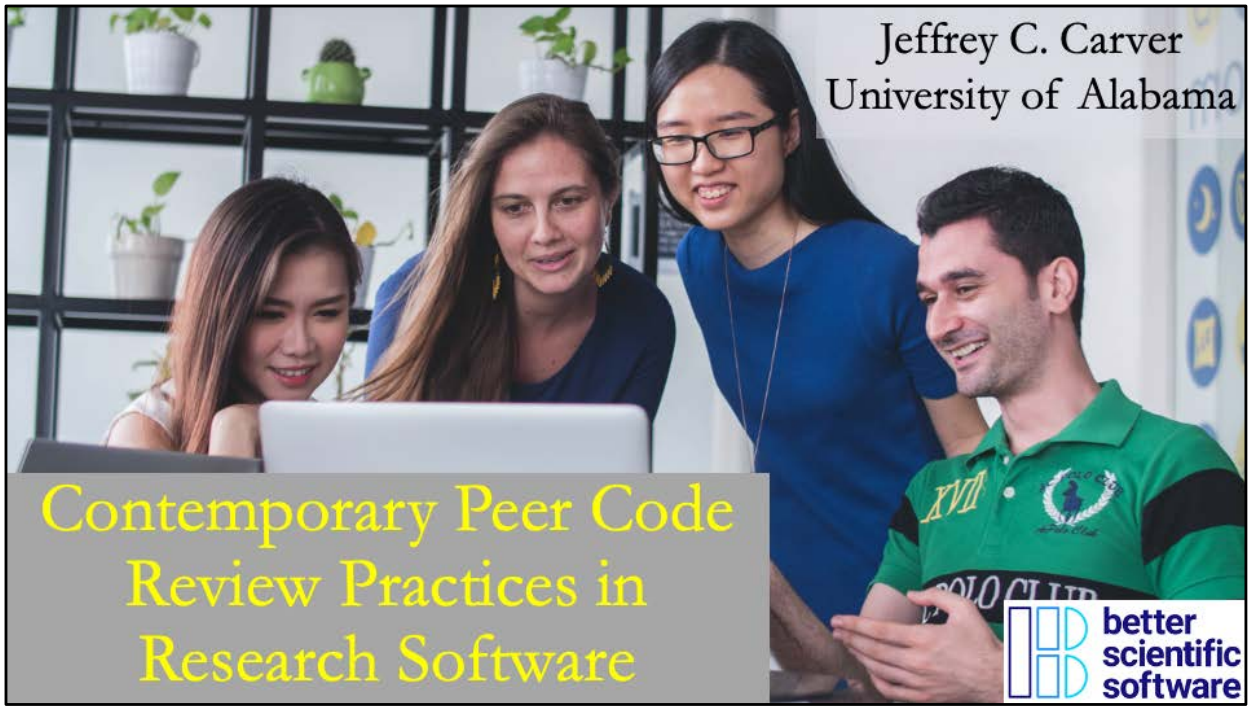
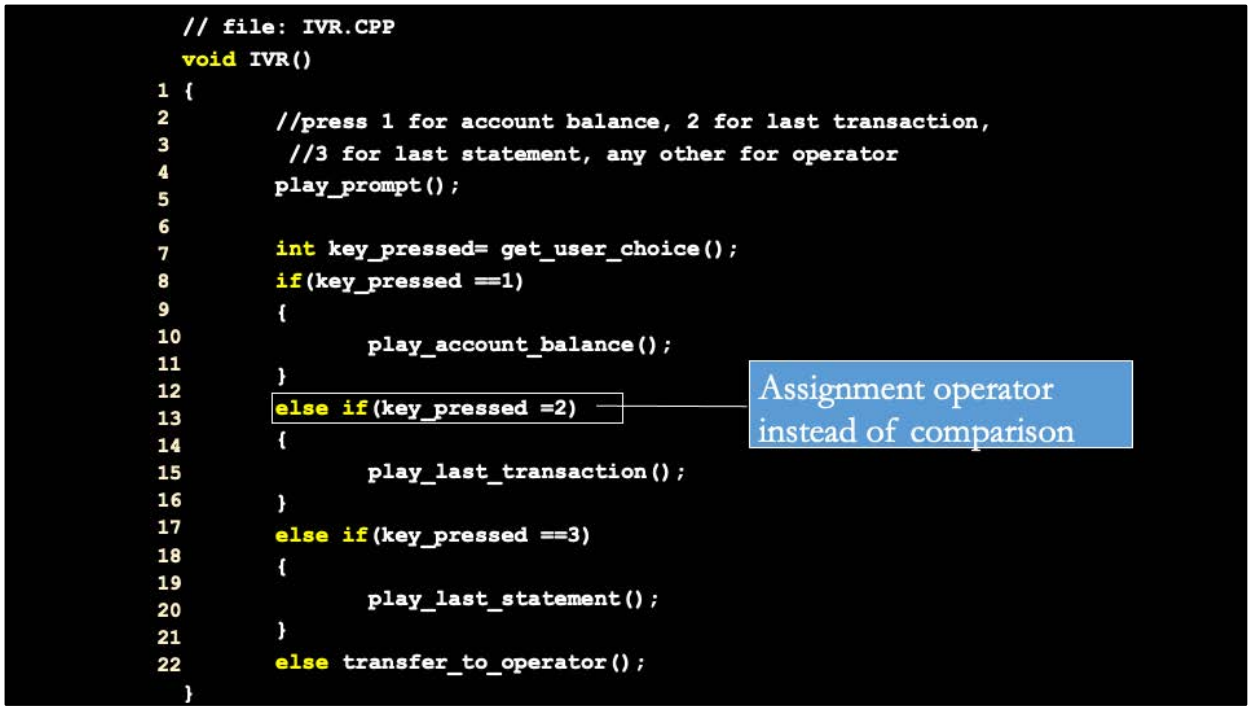


- Average defect detection rates
  - Unit testing – 25%
  - Function testing – 35%
  - Integration testing – 45%

- Average effectiveness of design/code inspections – 55%
- JPL estimates that it saves about $25,000 per inspection by finding and fixing defects at an early stage

All data from *Code complete*

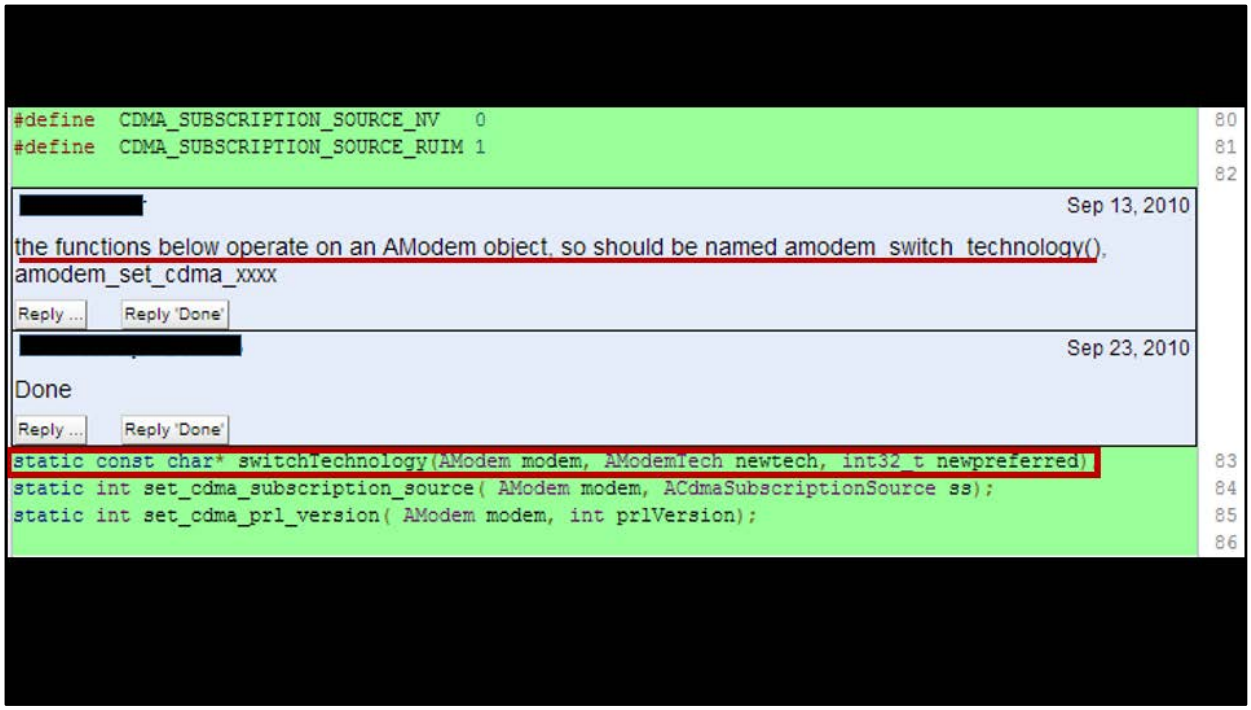


Please check this small C++ function. There is a mistake in the code. Can anyone please identify the mistake?

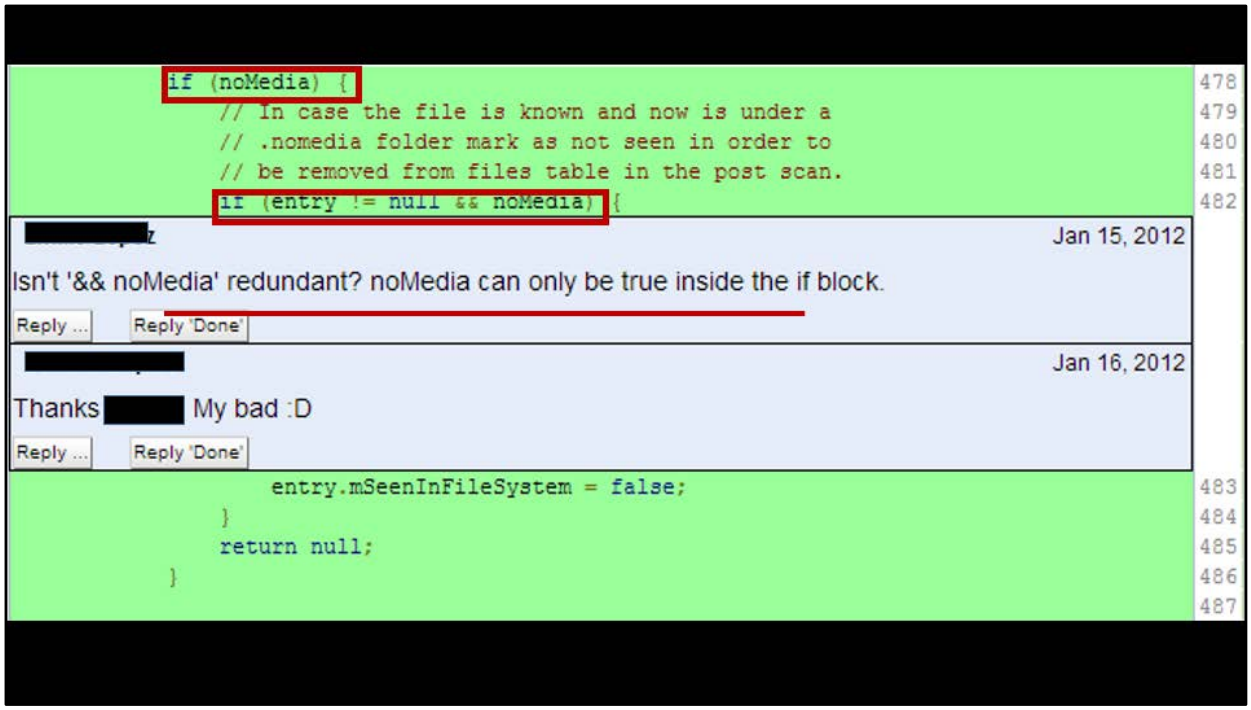
On line 13, assignment operator instead of comparison. This types are mistakes are sometimes hard to identify.


1 //file: printer.java
2 if (user.isAuthenticated)
3 {
4 userAccess = checkUserAuthorization(user);
5
6 //if user has access to printer
7 if(user.isAuthenticated && userAccess.printer)
8 printUsageReport ();
9 else
10 emailUsageReport();
11}
Redundant check

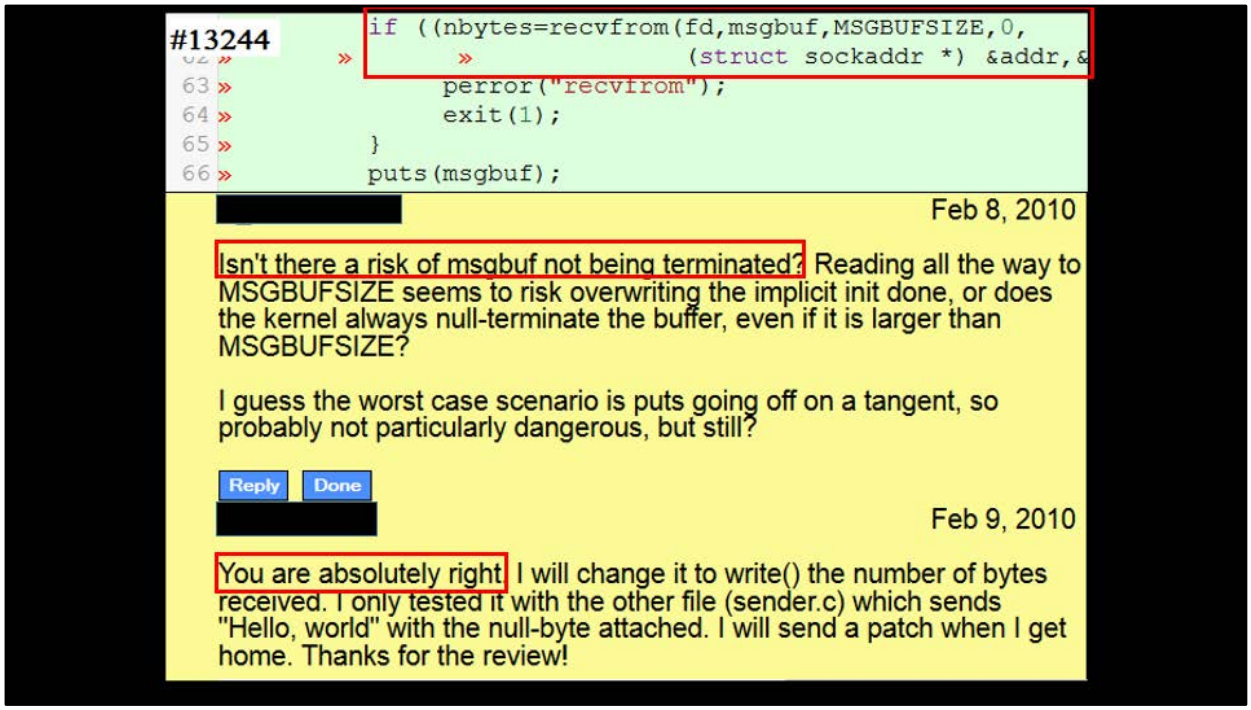
Do you think only novice developers make these mistakes?



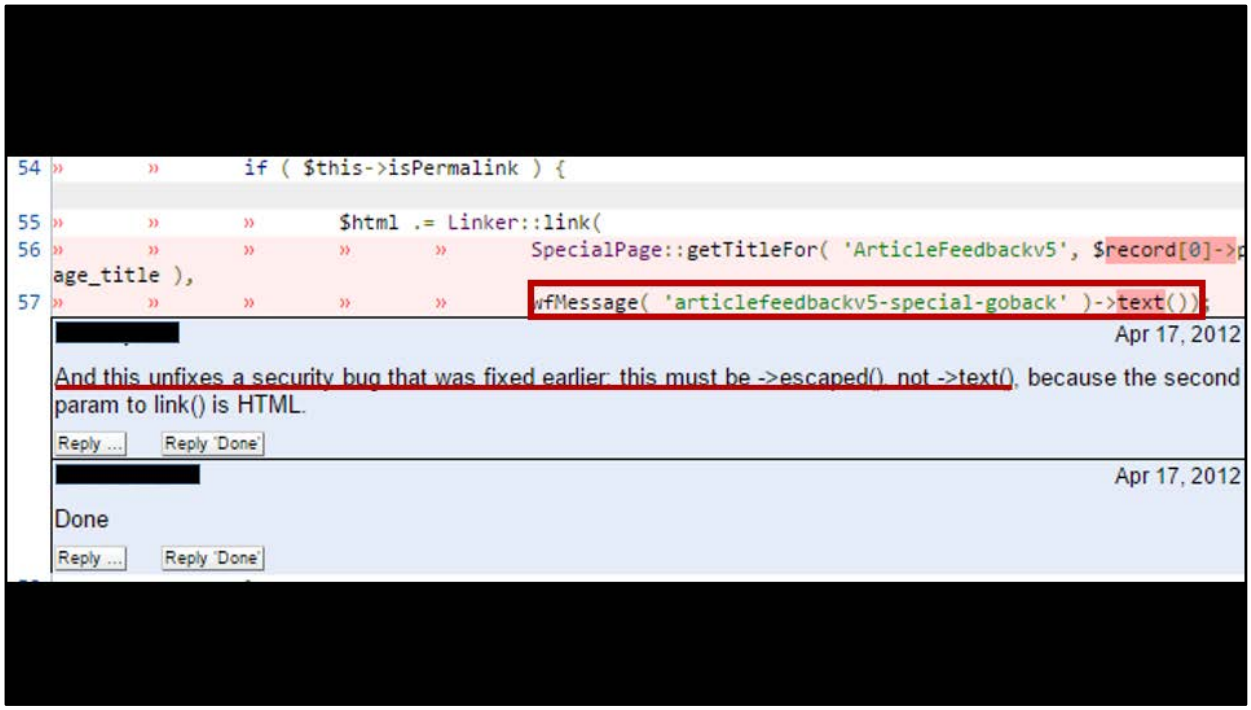
In the first example, the author has used camelCases in identifier naming, where other lines have used underscores. This is a style inconsistency issue identified by the reviewer.



Author created a silly mistake of including a redundant check. Many times developers commit this type silly mistakes. In this case, it was not a bug, but sometimes it can cause defects.

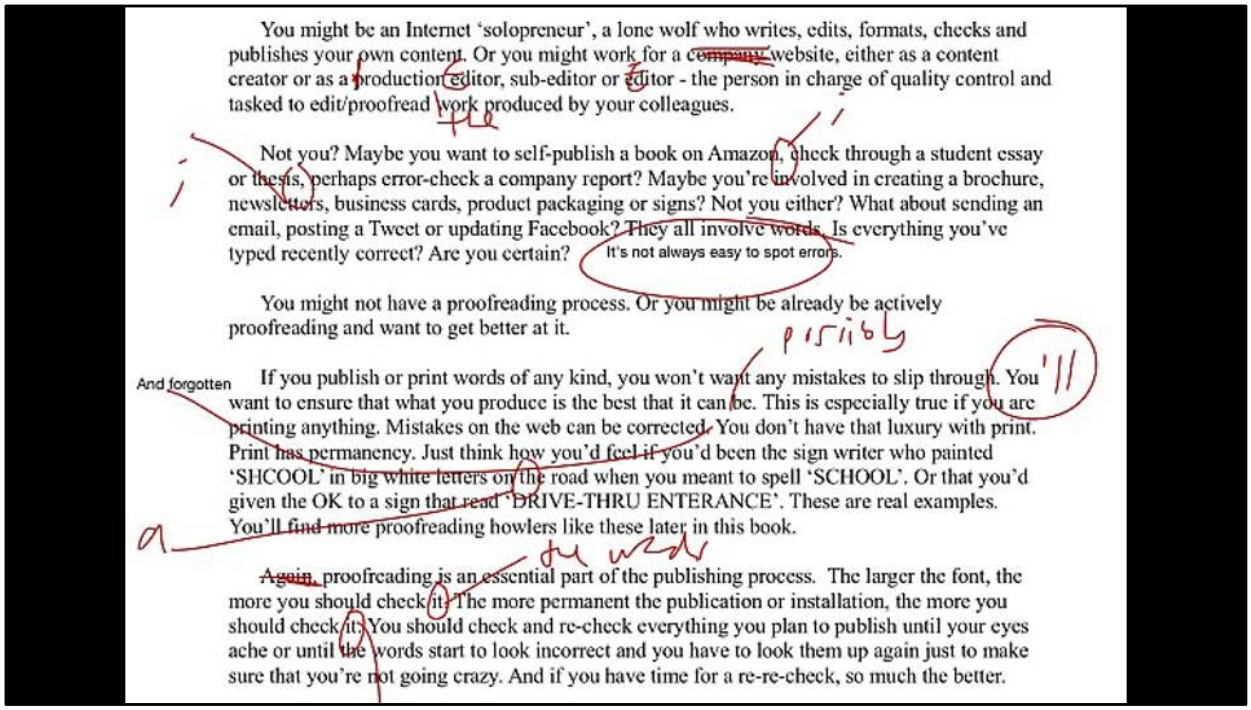


Buffer Overflow



In this example, the author has used a unsafe html handling method. This has the potential to open cross site scripting attack. The reviewer identifies that and the author fixes it.

We all make mistakes. Even quite experienced programmers still make mistakes. Sometimes we are unable to identify the mistake even it is right in front of our eyes. Therefore, we require some quality assurance checks to eliminate defects.



You might be an Internet 'solopreneur', a lone wolf who writes, edits, formats, checks and publishes your own content. Or you might work for a company website, either as a content creator or as a production editor, sub-editor or editor - the person in charge of quality control and tasked to edit/proofread work produced by your colleagues.

Not you? Maybe you want to self-publish a book on Amazon, check through a student essay or thesis, perhaps error-check a company report? Maybe you're involved in creating a brochure, newsletters, business cards, product packaging or signs? Not you either? What about sending an email, posting a Tweet or updating Facebook? They all involve words. Is everything you've typed recently correct? Are you certain? It's not always easy to spot errors.

You might not have a proofreading process. Or you might be already be actively proofreading and want to get better at it.

And forgotten If you publish or print words of any kind, you won't want any mistakes to slip through. You want to ensure that what you produce is the best that it can be. This is especially true if you are printing anything. Mistakes on the web can be corrected. You don't have that luxury with print. Print has permanency. Just think how you'd feel if you'd been the sign writer who painted 'SHCOOL' in big white letters on the road when you meant to spell 'SCHOOL'. Or that you'd given the OK to a sign that read 'DRIVE-THRU ENTERANCE'. These are real examples. You'll find more proofreading howlers like these later in this book.

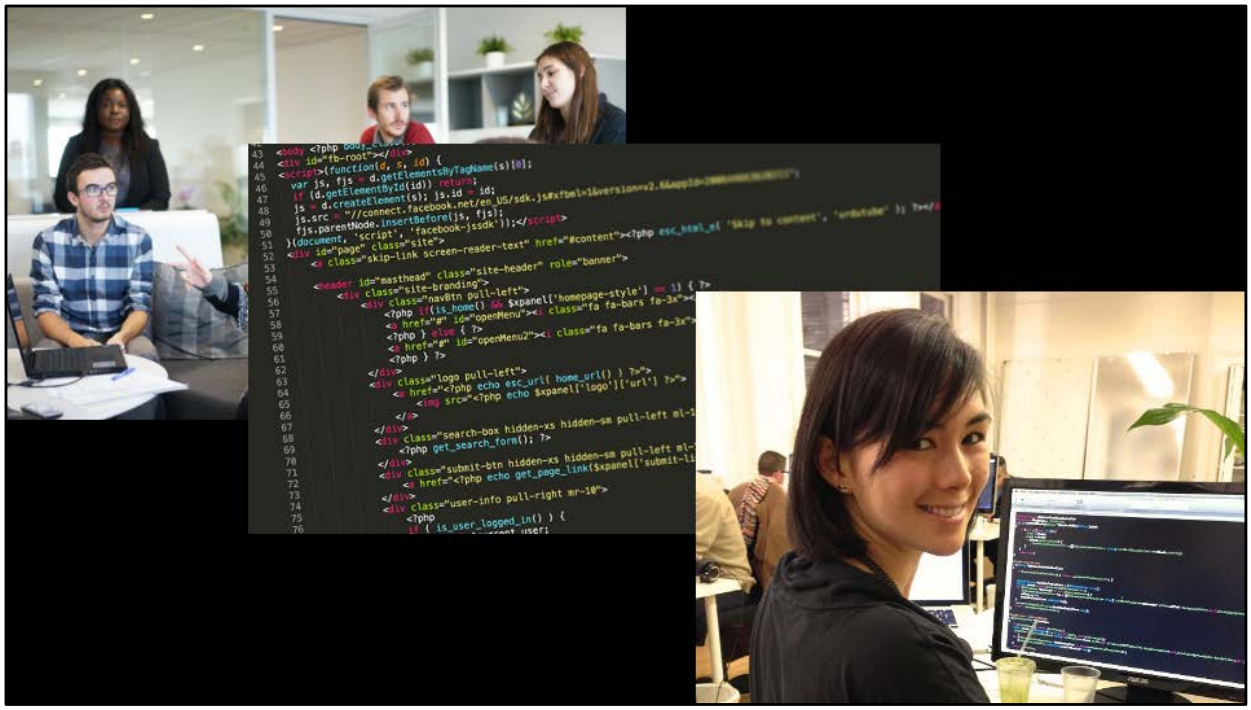
Again, proofreading is an essential part of the publishing process. The larger the font, the more you should check it. The more permanent the publication or installation, the more you should check it. You should check and re-check everything you plan to publish until your eyes ache or until the words start to look incorrect and you have to look them up again just to make sure that you're not going crazy. And if you have time for a re-re-check, so much the better.

For example, book publishers face similar problem in correcting text. For years, they have employed proofreaders, who can examine the text with a fresh pair of eyes. You may also have the experience of someone else read your writing and returning you something like this.

Software development has a similar practice as proofreading, which we call peer code review. Peer code review is the process of analyzing code written by a teammate to judge whether it is of sufficient quality to be integrated into the main project codebase. Today, I will talk about contemporary peer code reviews.

If we are going to do code review, we have to sacrifice something?
What gets sacrificed?
Is the tradeoff worth it?


Code Review Goals



- Team building
  - Better shared understanding
  - Team cohesion
  - Peer impression

- Code Quality
  - Find/fix defects early
  - Identify common problems
  - Different perspectives
  - Consistency in code/design
  - More maintainable code

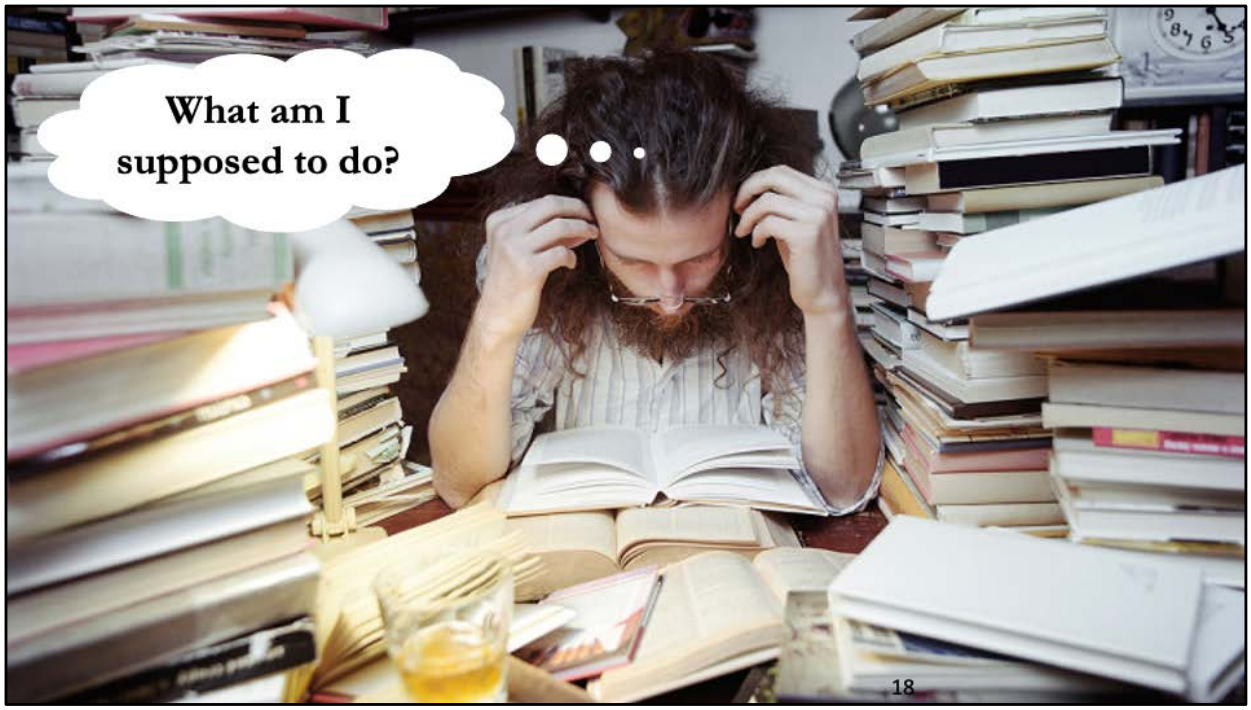
- Personal
  - Learning

The goal is for peace and harmony in the team, not antagonism

Code Review Practices

- Code review can come to nothing or harm the interpersonal relationships when they are done wrong.
- Hence it is important to pay attention to the human aspects of code review

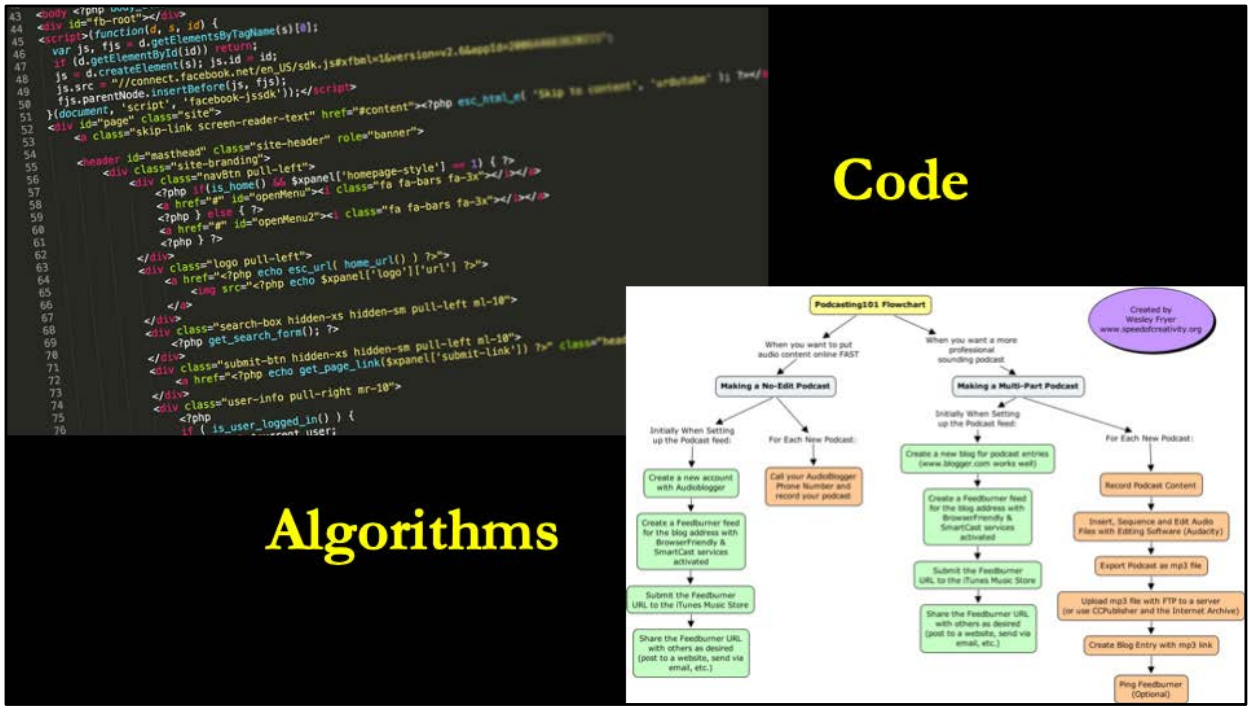
- Realize that the goal of code review it to improve the overall code, not to evaluate the quality or worth of the developer
- Remove the fear of making to mistakes an create an atmosphere where admitting and fixing is OK
- You are not your code
- Be humble
    - You will make mistakes, we all do
    - Someone else will always know more, its ok, learn from them
    - People bring different perspectives, that's a good thing
- Fight for what you believe, but gracefully accept defeat



- Focus on the code not the author
  - Use “I” statements rather than “you” statements
  - Criticize the author’s behavior, not their attributes
  - Talk about the code, not the coder
- Ask questions rather than make statements – avoid “why” questions
- Accept that there are different solutions
- Choose carefully which battles to fight
- Remember to praise good code
- Take your time and do it well

Code Review Techniques

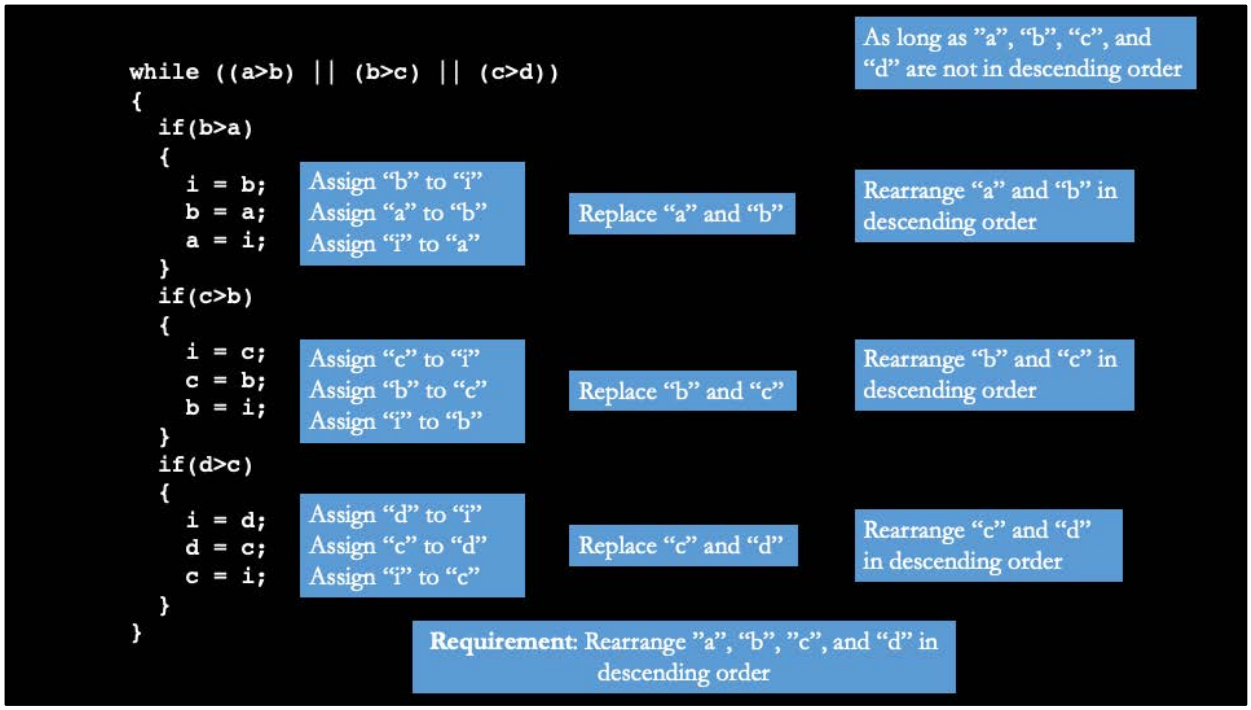
- We want to focus on what people are good at and let computers do what they are good at



- Examine the code
  - Is the code readable to a human?
  - Are variables and method names clear?
  - Is there sufficient documentation for someone to come back 6 months later (or someone new) to understand what the code is doing?
- Examine the algorithms in detail
  - Are there any hidden assumptions, not specified, that could cause problems?
  - Are there edge cases that may not work?
  - What happens with bad or missing data?
  - Does the algorithm do what it is supposed to? – Use stepwise abstraction

Example - Stepwise Abstraction

- Examine the algorithm embedded in the code
- Start at the bottom, extract low-level functionality
- Group low-level functionality into higher-level
- At top level, compare with desired plan

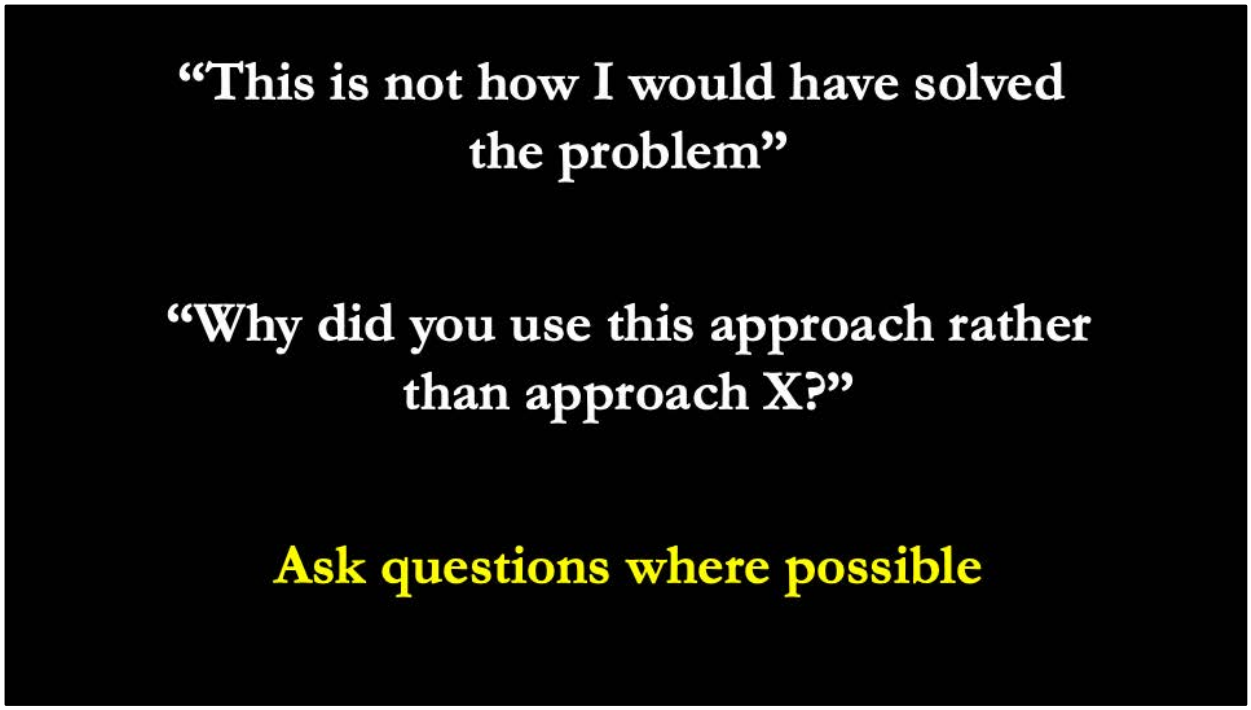

while ((a>b) || (b>c) || (c>d))
{
if(b>a)
{
i = b;
b = a;
a = i;
}
if(c>b)
{
i = c;
c = b;
b = i;
}
if(d>c)
{
i = d;
d = c;
c = i;
}
}
As long as "a", "b", "c", and "d" are not in descending order
Assign "b" to "i"
Assign "a" to "b"
Assign "i" to "a"
Replace "a" and "b"
Rearrange "a" and "b" in descending order
Assign "c" to "i"
Assign "b" to "c"
Assign "i" to "b"
Replace "b" and "c"
Rearrange "b" and "c" in descending order
Assign "d" to "i"
Assign "c" to "d"
Assign "i" to "c"
Replace "c" and "d"
Rearrange "c" and "d" in descending order
Requirement: Rearrange "a", "b", "c", and "d" in descending order

Code Review Comment Exercise

"You are writing cryptic code"

"Its hard for me to grasp what is going on in the code"

Use I-Messages



- Mention blog post here, the idea is that the question should suggest a solution, but still be a question.

"You are sloppy when it comes to writing tests"

"I believe that you should pay more attention to writing tests"

Criticize the author's behavior, not the author

"You're requesting the serve multiple times, which is inefficient"

"This code is requesting the service multiple times, which is inefficient"

Talk about the code, not the coder

"I always use fixed timestamps in tests and you should too"

"I would always use fixed timestamps in tests for better reproducibility, but in this simple test, using the current timestamp is also ok"

Accept different solutions


Developer
Checklist
My code compiles
My code has been tested and has unit tests
My code includes appropriate comments
My code is tidy / follows coding standard
I have documented corner cases
I have documented workarounds
…


Reviewer
Checklist
Comments are understandable and appropriate
Comments are neither too many or too few
Exceptions are appropriately handled
Repetitive code has been factored out
Frameworks have been used appropriately
Functionality fits the design/architecture
Code is testable
Code compiles

- Practice lightweight code reviews.
- Review fewer than 400 lines of code at a time.
- Inspection rate should be under 500 LOC per hour.
- Do not review for more than 60 minutes at a time.
- Set goals and capture metrics.
- Authors should annotate source code before review.
- Use checklists.

- Establish a process for fixing defects found.

- Foster a positive code review culture.

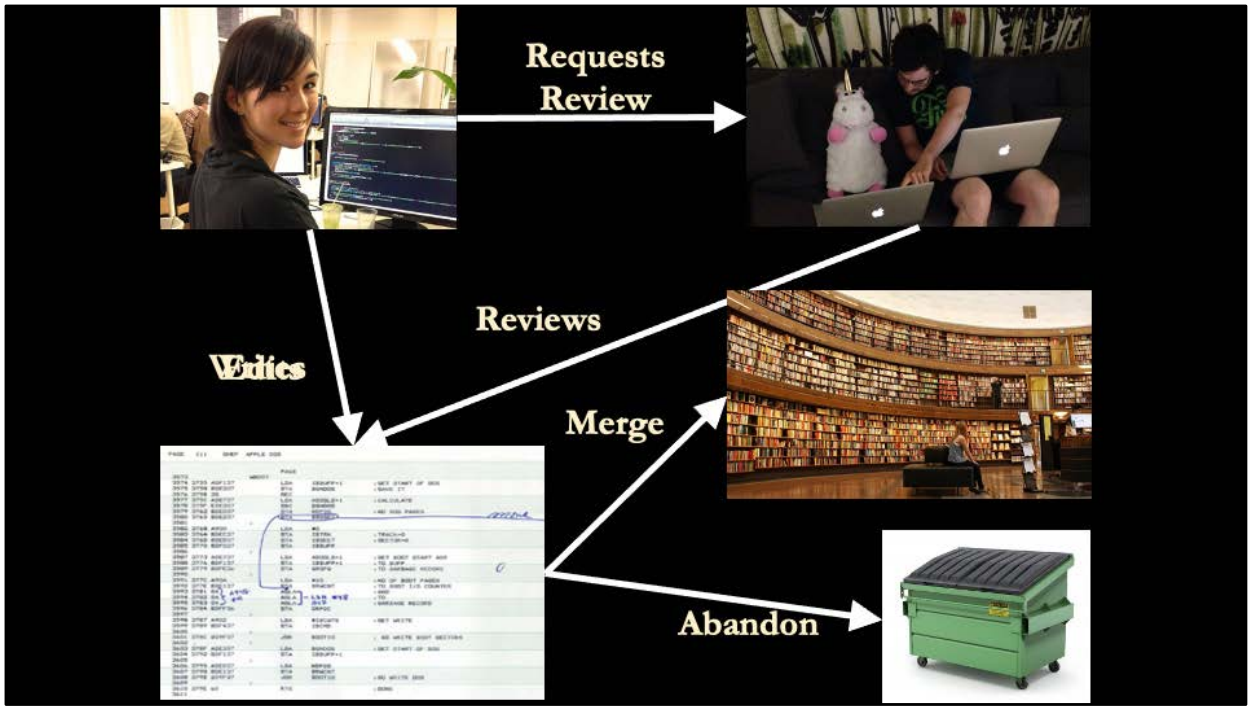
- Embrace the subconscious implications of peer review.

Issues Identified during code reviews

- Misunderstood requirements
- Project design violations
- Coding style
- Critical security defects
- Unsafe methods
- Inefficient code
- Malicious code
- Inadequate input validation
- Lack of exception handling

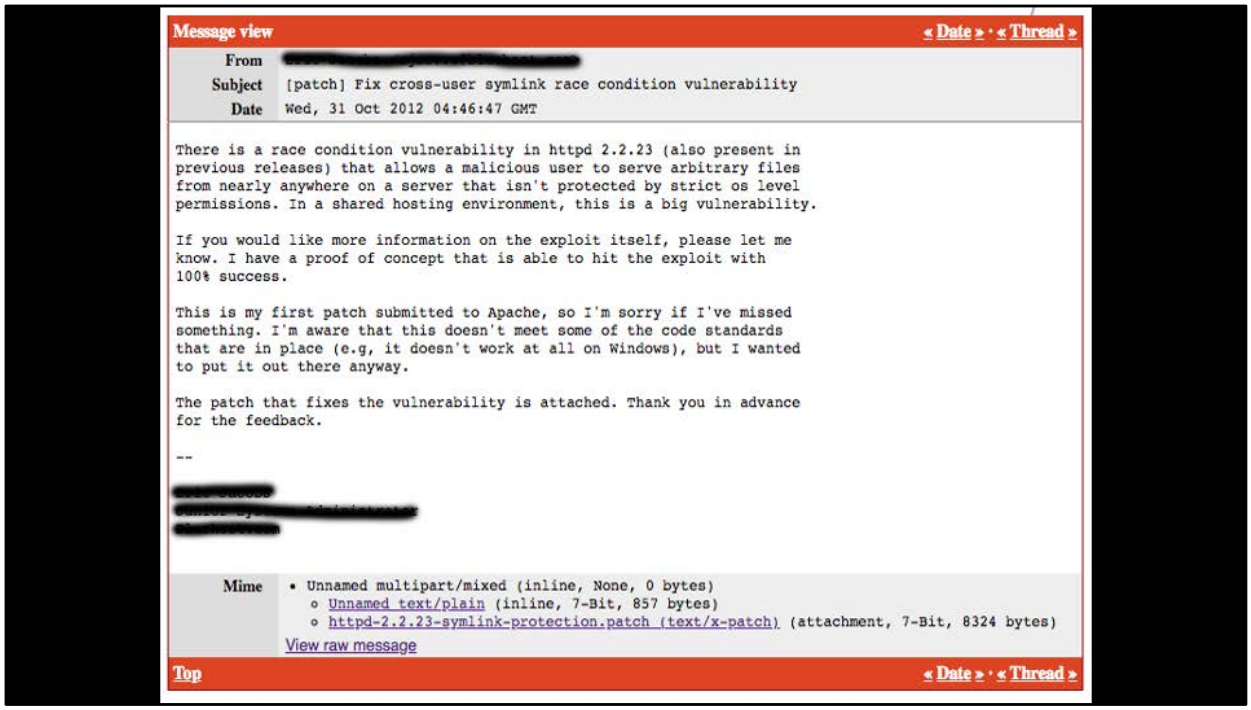
- Cultural difference between scientific community and software engineering community
- Correct results are unknown in many cases
- Testing is extensively complex in scientific software
- Common testing approaches may not fit
- May be better to review the scientific algorithm than to extensively test code
- Lack of proper testing knowledge
- Test to check the science, not the software
- Tend to test when development is about to finish

Typical Code Review Workflow



Let’s look into a typical code review workflow. Code review process starts after an author writes a code and submits the code for review by creating a review request. A reviewer accepts the review request. He reviews the code and provides feedback. The author modifies the code to address the review comments and submits the code again for review. This process of author modifying code and reviewer providing suggestions repeats until the reviewer approves the code or the author abandons the change. If the code is approved the author can integrate the code into main project code base.

Mailing List Code Review



Message view « Date » · « Thread »

From [redacted]
Subject [patch] Fix cross-user symlink race condition vulnerability
Date Wed, 31 Oct 2012 04:46:47 GMT

There is a race condition vulnerability in httpd 2.2.23 (also present in previous releases) that allows a malicious user to serve arbitrary files from nearly anywhere on a server that isn't protected by strict os level permissions. In a shared hosting environment, this is a big vulnerability.

If you would like more information on the exploit itself, please let me know. I have a proof of concept that is able to hit the exploit with 100% success.

This is my first patch submitted to Apache, so I'm sorry if I've missed something. I'm aware that this doesn't meet some of the code standards that are in place (e.g, it doesn't work at all on Windows), but I wanted to put it out there anyway.

The patch that fixes the vulnerability is attached. Thank you in advance for the feedback.

--

[redacted]

Mime
- Unnamed multipart/mixed (inline, None, 0 bytes)
  - Unnamed text/plain (inline, 7-Bit, 857 bytes)
  - httpd-2.2.23-symlink-protection.patch (text/x-patch) (attachment, 7-Bit, 8324 bytes)

View raw message

Top « Date » · « Thread »

- Generate diff file
- Email to list with pre-determined keywords
- Hope someone responds

- Problems
  - Lack of code context
  - Code integration
  - Review request management
  - Ignored requests
  - Reviewing large patches

Pull Requests

Tool-Based Code Reviews

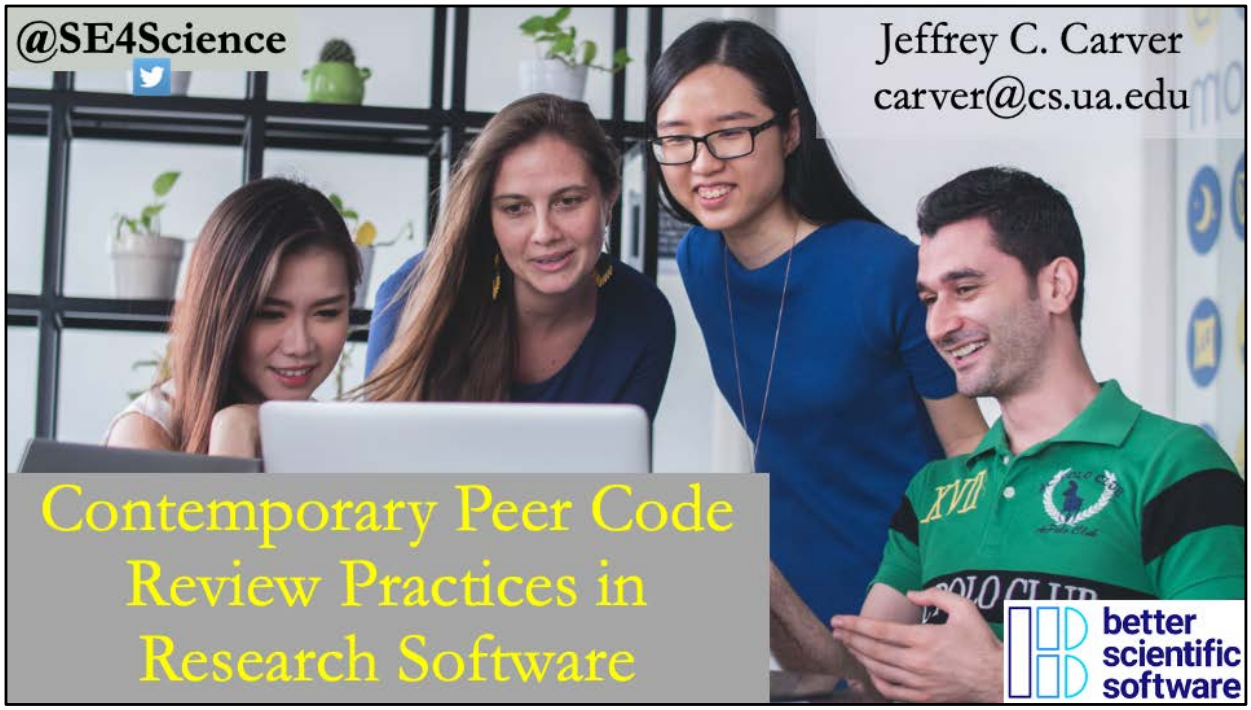
- Informal
- Asynchronous
- Tool-based
- High adoption
- ~6 hours/week

Code Review Tools

Contemporary code reviews are tool-based. Popular code review tools include: Gerrit, ReviewBoard, Phabricator, and Crucible.

References for further reading

- *Code Complete*, by Steve McConnel
- https://www.codeproject.com/articles/524235/codeplus reviewplusguidelines
- https://blog.philipphauer.de/code-review-guidelines
- https://github.com/joho/awesome-code-review
- https://www.planetgeek.ch/wp-content/uploads/2013/06/Clean-Code-V2.1.pdf



- Average defect detection rates
  - Unit testing – 25%
  - Function testing – 35%
  - Integration testing – 45%

- Average effectiveness of design/code inspections – 55%
- JPL estimates that it saves about $25,000 per inspection by finding and fixing defects at an early stage

All data from *Code complete*

Photo Credits

- http://incolors.club/collectionfdwn-female-computer-programmer.htm
- http://tech.trivago.com/img/posts/code-review/code-review-3.jpg
- http://www.protectitip.com/wp-content/uploads/2014/11/Software-Code.jpg
- http://www.computerhistory.org/atchm/wp-content/uploads/2013/11/marked_up_listing-542x404.jpg
- https://static1.squarespace.com/static/53798babe4b0fca9449cf693/t/53f78774e4b0ce4d05e4152f/1408730997720/
- https://residentialwastesystems.com/wp-content/uploads/2016/10/dumpsters-trumbull-ct.jpg
- http://www.hipaasecurenow.com/index.php/beckers-hipaa-compliance-8-best-practices/
- https://commons.wikimedia.org/wiki/File:Collaboration_(9601759166).jpg#metadata
- http://entertainment.time.com/2012/05/09/confessions-of-another-book-reviewer/
- Photo by Mimi Thian on Unsplash [Slide 1]