“The competent programmer is fully aware of the strictly limited size of his own skull; therefore he approaches the programming task in full humility, and among other things he avoids clever tricks like the plague.

– Edsger Dijkstra
Peer Reviews

- Anti-Patterns:
  - No peer reviews
  - Reviews too informal/too fast
  - Reviews find <50% of all bugs

- Fresh eyes find defects
  - Code and other document benefit from a second (and third) set of eyes
  - Peer reviews find more bugs/$ than testing
    - And, they find them earlier when bugs are cheaper to fix
  - Everything written down can benefit from a review
Most Effective Quality Practices


Ranked by defect removal effectiveness in percent defects detectable at that stage that are removed. “*” means exceptionally productive technique (more than 750+ function points/month)

- * 87% static code analysis ("lint" tools, compiler warnings)
- 85% design inspection
- 85% code inspection
- 82% Quality Function Deployment (requirements analysis)
- 80% test plan inspection
- 78% test script inspection
- * 77% document review (other documents)
- 75% pair programming (informal on-the-fly review)
- 70% bug repair inspection
- * 65% usability testing
- 50% subroutine testing (unit test)
- * 45% SQA (Software Quality Assurance) review
- * 40% acceptance testing
Peer Reviews Are Effective + Efficient

Defect Removal by Phase - Typical Project from 5 years earlier

- No reviews, no unit test, no integration test, ...
- Most bugs found in system test!

5 years later...

- Found more bugs total
- Almost no bugs left in system test!

- Found many bugs up front, where fixes are cheaper

[Source: Roger G., Aug. 2005]
Gold Standard: Fagan Style Inspections

- Methodical, in-person review meetings
  - Pre-meeting familiarity with project
  - Producer explains item then leaves
  - Moderator keeps things moving
  - Reader (not author) summarizes as you go
  - Reviewers go over every line, using checklists (perspective-based)
  - Recorder takes written notes
  - Result: written list of defects. The Producer fixes code off-line
  - Re-inspection if the defect rate was too high

- Methodical reviews are the most cost effective
  - Important to measure bug discovery rate to ensure review quality
Rules for Successful Peer Reviews

- Inspect the item, not the author
  - Don’t attack the author.

- Don’t get defensive
  - Nobody writes perfect code. Get over it.

- Find but don’t fix problems
  - Don’t try to fix them; just identify them.

- Limit meetings to two hours
  - People are less productive after that point.

- Keep a reasonable pace
  - About 150 lines of code (or equivalent) per hour. Too fast and too slow are both bad.

- Avoid “religious” debates on style
  - Enforce conformance to your style guide. No debates on whether style guide is correct.

- Inspect, early, often, and as formally as you can
  - Keep records to document value (might take a while to mature).
# Example Light-Weight Review Report

<table>
<thead>
<tr>
<th>Date:</th>
<th>4/17/2011</th>
</tr>
</thead>
<tbody>
<tr>
<td>Artifact:</td>
<td>Xyzzy.cpp  Functions: Foo(), Bar(), Baz()</td>
</tr>
<tr>
<td>Reviewers:</td>
<td>Stella K., Joe B., Sam Q., Trish R.</td>
</tr>
<tr>
<td>Size:</td>
<td>357 SLOC</td>
</tr>
<tr>
<td>Time Spent:</td>
<td>112 Minutes</td>
</tr>
<tr>
<td># Issues:</td>
<td>3</td>
</tr>
<tr>
<td>Outcome:</td>
<td>Re-Review of Bug Fixes Required</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Issue#</th>
<th>Issue Description</th>
<th>Status</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Issue 1...</td>
<td>Fixed</td>
</tr>
<tr>
<td>2</td>
<td>Issue 2...</td>
<td>Bugzilla</td>
</tr>
<tr>
<td>3</td>
<td>Issue 3...</td>
<td>Bugzilla</td>
</tr>
<tr>
<td>4</td>
<td>Issue 4...</td>
<td>Not a Bug</td>
</tr>
<tr>
<td>5</td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td></td>
<td></td>
</tr>
<tr>
<td>8</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Status Key:**
- Fixed (trivial fix by author; no need to enter in defect list)
- Bugzilla (entered into project defect system)
- Not a Bug (false alarm)
Perspective-Based Peer Reviews

Perspective-based Peer Reviews are 35% more effective
[https://www.cs.umd.edu/projects/SoftEng/ESEG/papers/82.78.pdf]

Mechanics of a Perspective-based review

- Divide a peer review checklist into three sections
- Assign each participant a different section of the checklist
  - OK to notice other things, but primary responsibility is that section
  - Multiple sets of eyes + perspective breadth

Example perspectives for a review:

- Control flow issues
- Data handling issues
- Style issues
# Peer Review Checklist: Embedded C Code

## Before Review:

0. Code compiles clean with extensive warning checks (e.g. MISRA C rules)

## Reviewer #1:

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

## Reviewer #2:

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>8</td>
<td></td>
<td></td>
</tr>
<tr>
<td>9</td>
<td></td>
<td></td>
</tr>
<tr>
<td>10</td>
<td></td>
<td></td>
</tr>
<tr>
<td>11</td>
<td></td>
<td></td>
</tr>
<tr>
<td>12</td>
<td></td>
<td></td>
</tr>
<tr>
<td>13</td>
<td></td>
<td></td>
</tr>
<tr>
<td>14</td>
<td></td>
<td></td>
</tr>
<tr>
<td>15</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

## Reviewer #3:

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>16</td>
<td></td>
<td></td>
</tr>
<tr>
<td>17</td>
<td></td>
<td></td>
</tr>
<tr>
<td>18</td>
<td></td>
<td></td>
</tr>
<tr>
<td>19</td>
<td></td>
<td></td>
</tr>
<tr>
<td>20</td>
<td></td>
<td></td>
</tr>
<tr>
<td>21</td>
<td></td>
<td></td>
</tr>
<tr>
<td>22</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

## All Reviewers

23. Does the code match the detailed design (correct functionality)?

24. Is the code as simple, obvious, and easy to review as possible?

For TWO Reviewers assign items: Reviewer#1: 1-11; 23-24 Reviewer#2: 12-24
Before (Ineffective Reviews)

Average Median = 12.9
Average Mean = 13.4
With Weekly Defect Reporting

Average Median = 13.0
Average Mean = 13.8
Review More Than Just The Code

LEGEND:
Artifacts To Peer Review

Static Analysis
Economics Of Peer Review

- Peer reviews provide more eyeballs to find bugs in an affordable way
  - Good embedded coding rate is 1-2 lines of code/person-hr
    - (Across entire project, including reqts, test, etc.)
  - A person can review 50-100 times faster than they can write code
    - If you have 4 people reviewing, that is still >10x faster than writing!
  - How much does peer review cost?
    - 4 people * 100-200 lines of code reviewed per hour
    - E.g., 300 lines; 4 people; 2 hrs review+1 hr prep = 25 LOC/person-hr
  - Reviews are only about 5%-10% of your project cost

- Good peer reviews find at least half the bugs!
  - And they find them early, so total project cost can be reduced

- Why is it folks say they don’t have time to do peer reviews?
Peer Review Best Practices

Formal reviews (inspections) optimize bugs/\$

- Target 10% of project effort to find 50% of bugs
  - You can review 100x faster than write code; it’s cheap
- Review everything written down, not just code
- Use a perspective-based checklist to find more bugs

Review pitfalls

- If your reviews find <50% of defects, they are BROKEN
  - The 80/20 rule does NOT apply to review formality! Formal reviews are best.
  - You can’t review at end; need to review throughout project

Review tools

- On-line review tools are OK, but not a substitute for in-person meeting
- Static analysis tools are great – but not a review!
Your code looks like song lyrics written using only the stuff that comes after the question mark in a URL.

It's like a JSON table of model numbers for flashlights with "tactical" in their names.

Like you read Turing's 1936 paper on computing and a page of JavaScript example code and guessed at everything in between.

It's like a Leet-speak translation of a manifesto by a survivalist cult leader who's for some reason obsessed with memory allocation.

I can get someone else to review my code. Not more than once, I bet.