Chapter 15: Review Techniques

Subject: Software Engineering

Program: BTech Computer Science and Engineering

Duration: 1 Hour

The Most Cost-Effective Quality Control Method

Catching Bugs Before They Run

The Opening Hook

The 80% Question

The Scenario:

If there was a technique that could find **60-80%** of software defects before a single line of code is ever executed, would you use it?

The Technique:

That technique is Reviews—the systematic examination of software artifacts by human eyes.

The Reality:

Code reviews and formal inspections are the most cost-effective quality control methods software engineering.

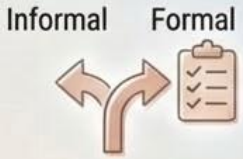
Today, we learn why they matter, how they work, and how to conduct them effectively.

Learning Objectives

By the end of this lecture, you will be able to:

Understand the Cost:
Explain the financial impact of software defects and the concept of defect amplification.

Analyze the Metrics:
Describe specific review metrics and analyze their true cost-effectiveness.



Classify the Types:
Clearly distinguish between informal reviews and formal technical reviews.

Execute the Process:
Describe the exact structure, conduct, and golden rules for executing Formal Technical Reviews (FTRs).

Part 1: The Economics of Defects

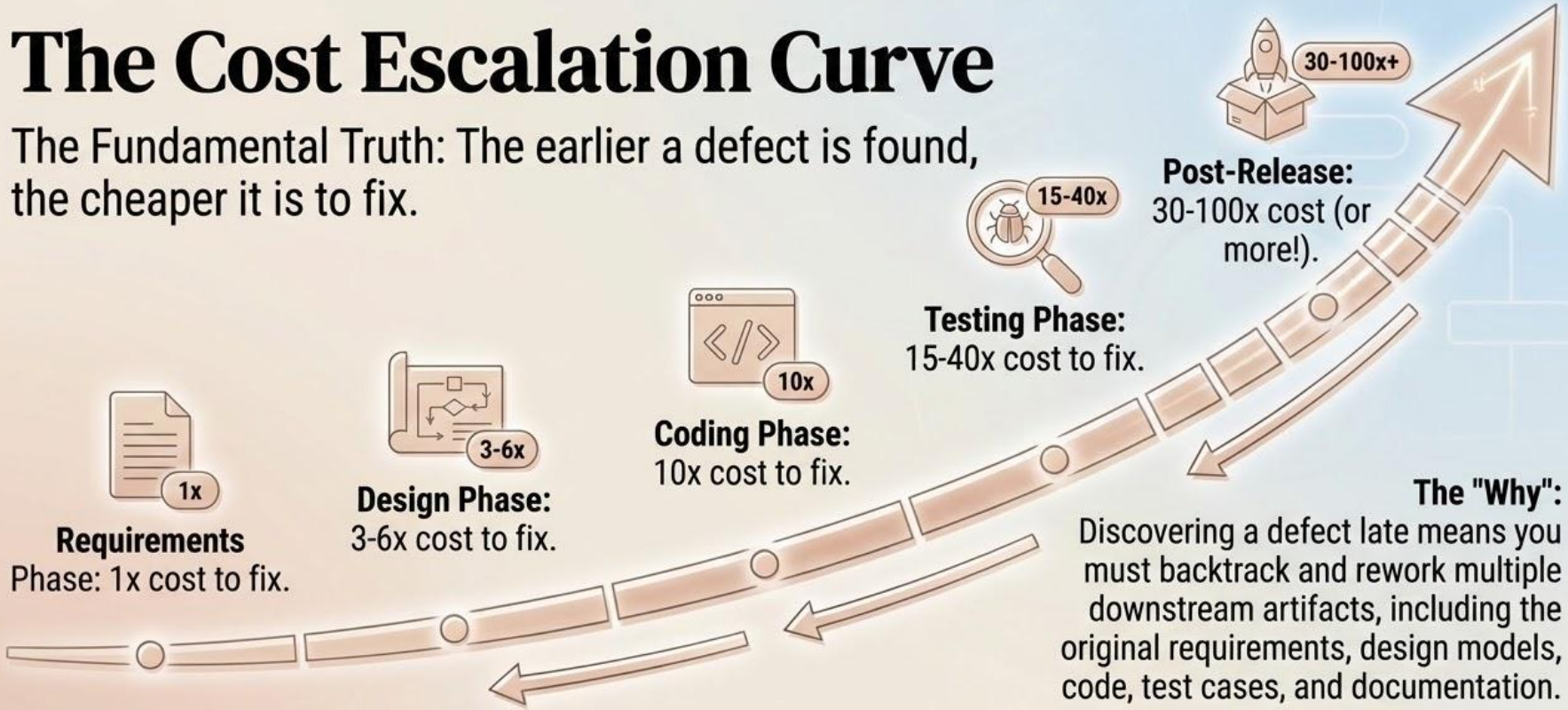
Sections 15.1 - 15.2

The Cost Impact of Software Defects

Focus: Why catching bugs early saves time, money, and project timelines.

The Cost Escalation Curve

The Fundamental Truth: The earlier a defect is found, the cheaper it is to fix.



Requirements Phase: 1x cost to fix.

Design Phase: 3-6x cost to fix.

Coding Phase: 10x cost to fix.

Testing Phase: 15-40x cost to fix.

Post-Release: 30-100x cost (or more!).

The "Why": Discovering a defect late means you must backtrack and rework multiple downstream artifacts, including the original requirements, design models, code, test cases, and documentation.

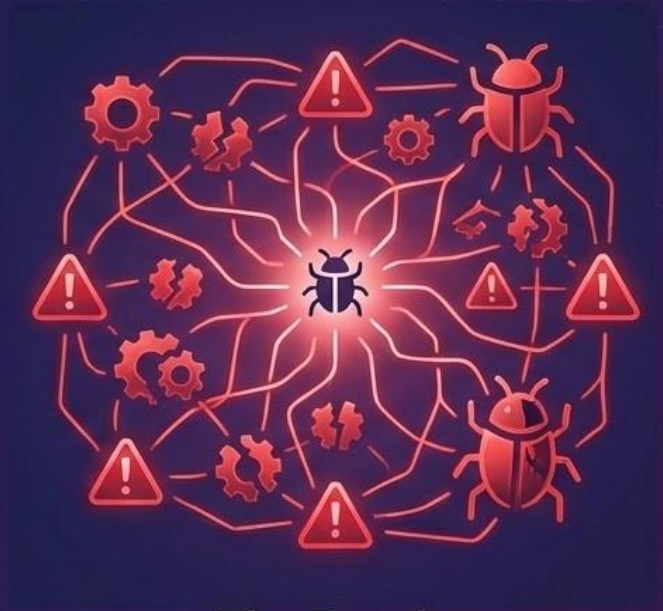
Slide 3: Defect Amplification

How Bugs Multiply

The Model:

- Defects introduced in one phase do not stay isolated.
- They “amplify” to create more defects in subsequent phases.

The Cascade Effect:



The Result:

One original defect quickly becomes a massive web of downstream failures.

Defect Removal Efficiency (DRE)

Measuring the Power of Reviews

The Metric:

Defect Removal Efficiency measures how effectively a team clears out bugs before shipping.

The Power of Reviews:

Conducting peer reviews catches defects exactly at the injection point (e.g., catching a requirements flaw during the requirements review).

The Formula:

$$\text{DRE} = \frac{\text{Defects found before release}}{\text{Total defects present}}$$

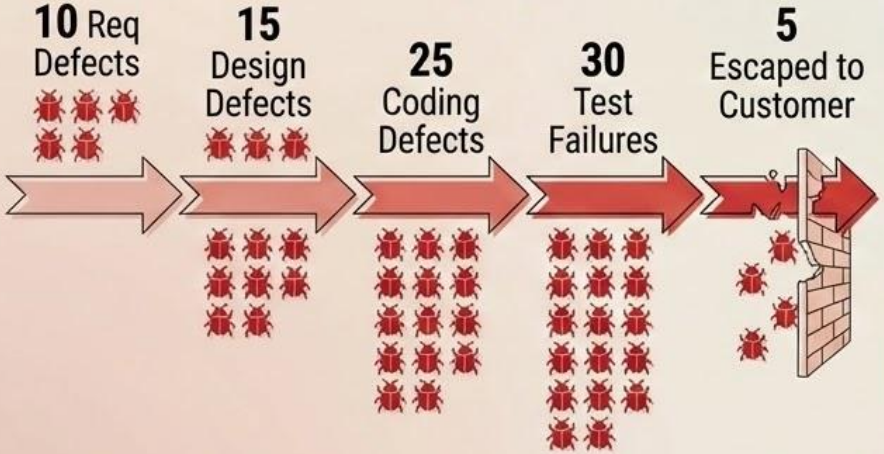
The Impact:

Catching it immediately prevents the defect amplifi-cattion cascade entirely.

The Impact of Reviews in Action

A Tale of Two Projects

Scenario A: Without Reviews
(The Cascade)



Scenario B: With Phase-by-Phase Reviews
(The Interception)

Phase	Requirements	Design	Code	Test	Release
Defects Injected	10	12	15	8	
Action	Req Review	Design Review	Code Review	Testing	3 Escaped
Defects Caught	(Caught 6)	(Caught 8)	(Caught 9)	(Caught 5)	

Measuring the Value of Reviews

Part 2: Review Metrics and Their Use

Sections 15.3.1 - 15.3.2

Focus:

How to track review effectiveness

and calculate their Return on Investment (ROI).

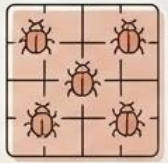
Analyzing Review Metrics

Section 15.3.1: You Cannot Manage What You Do Not Measure

To ensure reviews are actually working, teams must track specific data points.

Defects Found per Review Hour

Measures the overall efficiency of the review session.



Defect Density

Defects per unit size (e.g., per Function Point or per 1,000 Lines of Code [KLOC]). This is used to compare against organizational baselines.

Defects Found per Page/KLOC

Measures the baseline quality of the artifact being reviewed.

Review Rate

Pages or KLOC reviewed per hour.

Too fast → The team misses defects.

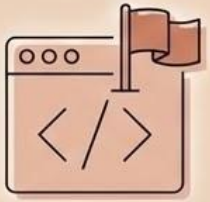
Target: ~4-8 pages/hour or ~150 LOC/hour for code.

Too slow → The process becomes inefficient.

Putting Metrics to Work

How to Apply the Data

Tracking numbers is only useful if it drives action. We use these metrics to:



Identify Problem Areas:

Flag specific modules or artifacts with unusually high defect density for total rewrites.

Recognize Talent:

Identify reviewers who consistently find more defects and leverage them for critical modules.

Calibrate Speed:

Adjust the team's review rate to find the "sweet spot" for optimal effectiveness without dragging out the schedule.

Cost-Effectiveness of Reviews

Section 15.3.2: The Real ROI

For every hour spent on reviews, how many hours of rework are saved? The numbers are striking:

Massive Time Savings:

Formal inspections can save **10 to 20 hours** of future rework for every **1 hour** invested in the review.

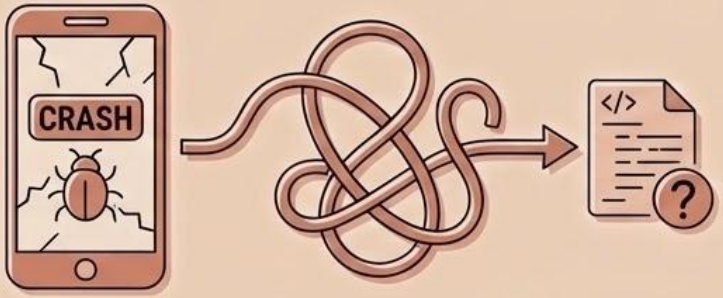
Better Than Testing:

Studies consistently show that reviews are **3 to 5 times more cost-effective** at finding defects than execution-based testing.

Why Do Reviews Beat Testing?

Source vs. Symptom

Testing: Finds the **symptom** of a bug



(e.g., the app crashes when you click "Submit"). The developer must then spend hours debugging to trace that symptom back to the source code flaw.

Reviews: Find the **defect** directly at the **source**

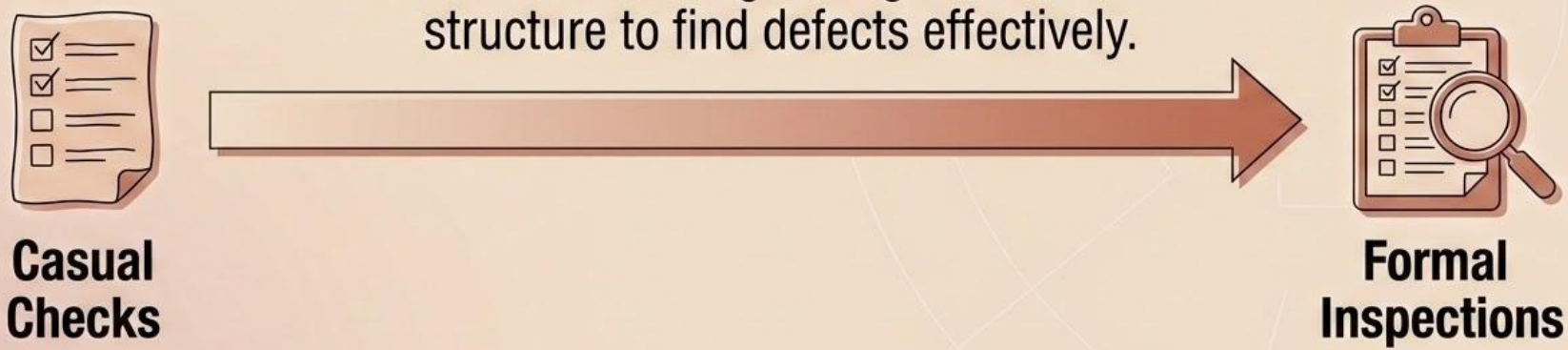
(e.g., a reviewer points at line 42 and says, "This logic fails if the user inputs a negative number"). There is zero time wasted on tracing or debugging.

Part 3: The Review Spectrum

Sections 15.4 - 15.6

From Casual Checks to Formal Inspections

- **Focus:** Choosing the right level of structure to find defects effectively.



The Formality Spectrum

Slide 2: Section 15.4: Matching the Method to the Need

Reviews range from casual conversations to highly structured, metric-driven meetings. You must choose the right level of formality for the specific context and risk level of the artifact.

Informal Reviews
(e.g., Pair Programming)

Walkthroughs

Technical Reviews

Inspections
(Most Formal)

Informal Reviews

Slide 3: Section 15.5: Fast and Flexible

Characteristics

- No formal process, agenda, or reporting.
- Can be as simple as asking a colleague, “Can you look at this code?”

Examples

- Desk checks
- Casual peer reviews
- Pair Programming (continuous, real-time informal review)

The Trade-offs

Advantages

- Fast, low overhead
- Highly effective at fostering team collaboration.

Disadvantages

- Inconsistent
- Generates no metrics for improvement
- Lacks the rigor needed to catch complex, deep-seated defects.

Formal Technical Reviews (FTR)

Slide 4: Section 15.6: The “Gold Standard”

What it is:

A structured, rigorous examination of a software artifact (code, design, requirements).

The Prime Directive:

The absolute objective is to find errors, ensure quality, and improve the product—NOT to evaluate or criticize the author.

The FTR Cast of Characters

Slide 5: Section 15.6.1: Defined Roles

A successful Formal Technical Review relies on specific people playing specific parts (usually 3-5 participants in total):

Producer (Author):
The person who created the work product. Their job is to present the material and answer questions.

Review Leader (Moderator):
The neutral facilitator. They plan the review, run the meeting, and ensure all guidelines are strictly followed.

Recorder (Scribe):
Takes detailed notes of all issues identified, decisions made, and action items assigned.

Reviewers:
Technical staff whose sole job is to examine the work product for defects prior to and during the meeting.

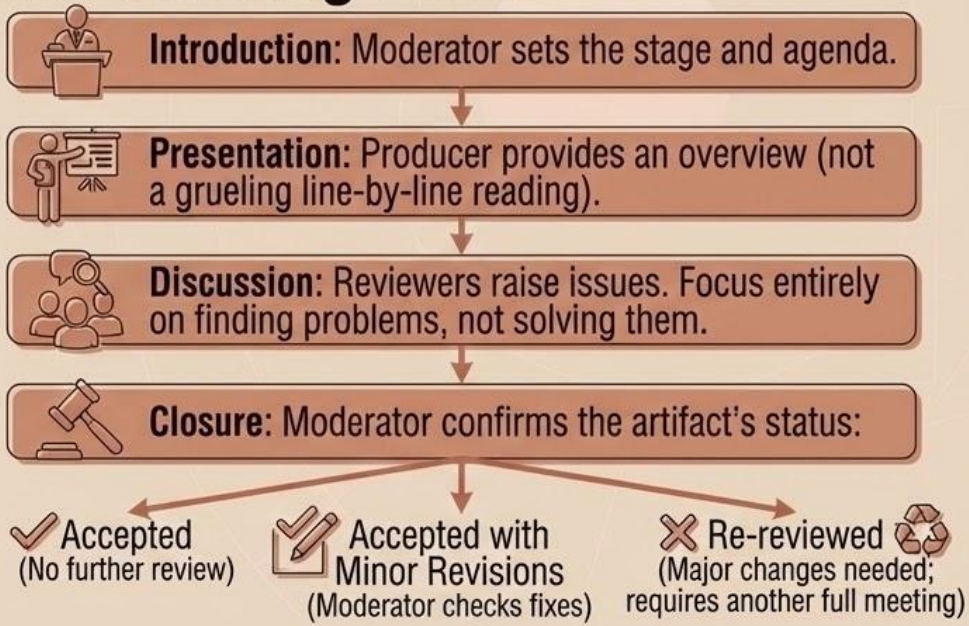
The FTR Meeting Structure

Slide 6: Section 15.6.1: Preparation and Execution

Pre-requisites:

- Artifact must be "complete enough."
- Reviewers must receive materials 2-3 days in advance and prepare a list of potential issues.
- Unprepared reviewers should not attend.

The Meeting Flow:



The FTR Reporting and Record Keeping

Slide 7: Section 15.6.2: Documenting the Outcome

If it isn't documented, it didn't happen. The Recorder generates:

Review Summary Report: What was reviewed, who attended, and the final outcome (Accept/Revise/Reject).

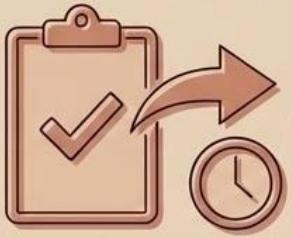
Issue Log: A detailed list of each defect including location, description, severity, and who is assigned to fix it.

Metrics Data: Effort spent (prep + meeting time) and the number/type of defects found. These feed directly into process improvement.

FTR Guidelines for Success

Slide 8: Section 15.6.3: The Golden Rules

Review the product, not the producer: Never criticize the author.



Note the issue, move on: Limit debate and problem-solving. Solving is for after the meeting.

Keep it under 2 hours: Attention spans fade; long reviews miss bugs.

Limit participants: 3 to 5 people is the ideal size.

Use checklists: Establish a checklist of common errors for each artifact type to guide reviewers.

Prepare: Preparation is mandatory, not optional.

Sample-Driven Reviews

Slide 9: Section 15.6.4: Scaling the Process

When to use:
For massive artifacts where a full review is practically impossible (e.g., thousands of pages of documentation).

How it works:

Select a statistically representative sample (e.g., every 10th page).

Conduct a thorough FTR on just that sample.

Analyze the defect density of the sample to infer the overall quality of the entire document.

Decision Rules:

Below threshold: Accept the whole artifact.

Above threshold: Reject whole artifact, require the author to rework it, and mandate a new sample review later.

Catching Mistakes Before They Catch Us

Synthesis of Software Reviews

Wrapping up the lecture

Reviews are Crucial:
Collaboration improves quality.

Follow the Golden Rules:
Keep it focused and efficient.

Use Checklists & Prepare:
Structure prevents oversights.

Scale with Sampling: Use data for large artifacts.

The Economics of Early Detection

Slide 2: The ROI of Reviews

The Exponential Cost

Defects cost exponentially more the later they are found.

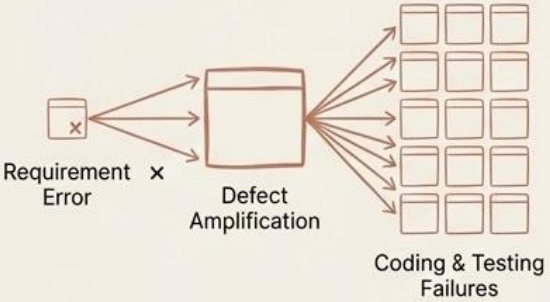
Left unchecked, defect amplification multiplies a single requirement error into dozens of coding and testing failures.

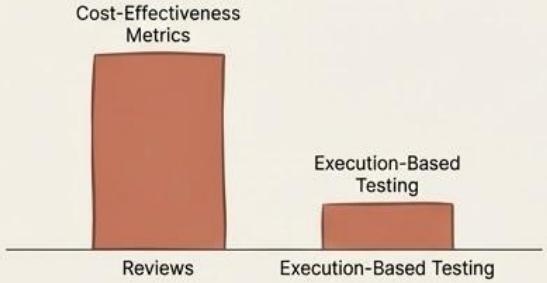
Unmatched Cost-Effectiveness

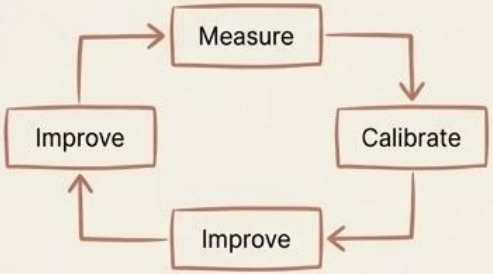
Reviews are the **most cost-effective defect detection** method available, consistently delivering **ROIs of 10:1 or higher** compared to execution-based testing.

Data-Driven Improvement

Utilizing **metrics—like defect density, review rate**, and **Defect Removal Efficiency (DRE)—** allows us to actively measure, calibrate, and improve the engineering process.



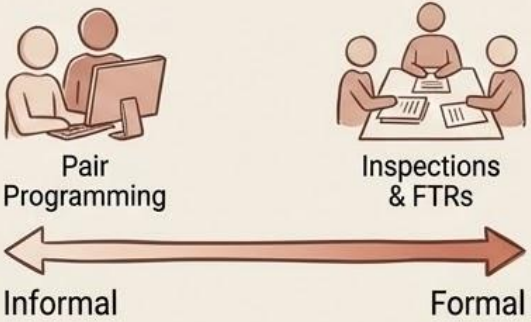




Structure and Execution

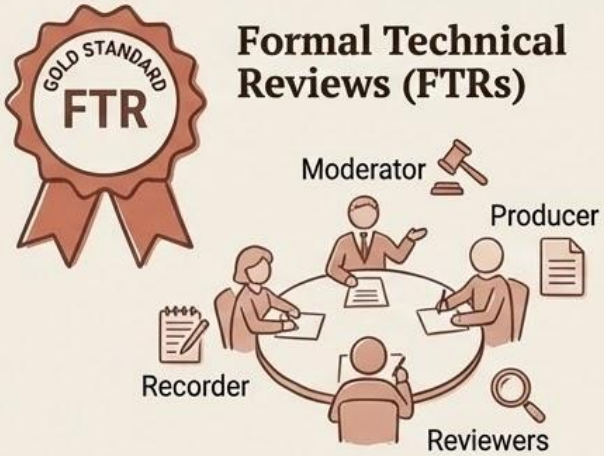
Slide 3: Right-Sizing the Process

The Formality Spectrum



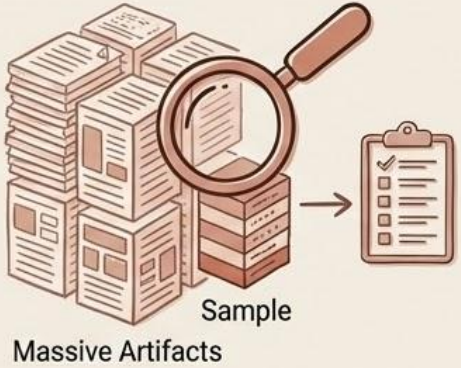
Reviews are not one-size-fits-all. They range from highly informal (pair programming) to highly formal (Inspections and FTRs).

Formal Technical Reviews (FTRs)



The gold standard requires a structured meeting, strict guidelines, and defined roles (Moderator, Producer, Recorder, Reviewers) to maximize defect finding while protecting egos.

Scaling Up

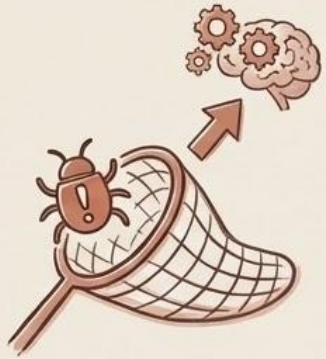


Sample-driven reviews provide a statistically sound, cost-effective way to assess massive artifacts where a 100% review is impossible.

Final Thought

Slide 4: Building a Mature Culture

> "Reviews are not about catching people making mistakes; they are about catching mistakes before they catch us. A mature engineering culture embraces reviews as a learning and improvement mechanism, not as a punishment."



The Core Philosophy:
Our goal is a collaborative pursuit of quality. Leave your ego at the door, focus on the artifact, and work as a team to build resilient software.