IMPLEMENTING A FORMAL TECHNICAL REVIEW PROCESS IN A SOFTWARE DEVELOPMENT ENVIRONMENT

Boniface C. Nwugwo
Engineering Services, ESPC
Eastman Kodak Company

OBJECTIVES

1. DEFINE TECHNICAL REVIEWS
   - **Definition of Reviews**
   - **Informal Reviews**
   - **Formal Reviews**
   - **Distinguish between Formal and Informal reviews**

2. EXPLAIN WHY WE NEED REVIEWS
   - **Illustration with Defect Amplification model**
   - **Illustration with Cost impact of early error detection**

3. HOW WE DID IT ON OUR PROJECT
   - **Our Software Group: who we are**
   - **Our Software Environment**
   - **Getting started**
   - **Sustaining momentum**
   - **Where we are now**

4. SOME REVIEW GUIDELINES
   - **Guidelines for doing reviews**

DEFINITION OF REVIEWS

A **Review** (any review) - a way of using the collective knowledge of a group of diverse people to:

1. Point out needed improvements in a product of a single person or team.

2. Confirm those parts of a product in which improvement is either not desired or not needed.

3. Achieve technical work of more uniform, or at least more predictable quality than can be achieved without reviews, in order to make technical work more manageable.

*4. Confirm traceability of implementation to design and requirements specifications.

...Pressman [PRE87]

* Boni's addition to the definition.

WHAT IS AN INFORMAL REVIEW?

- Informal Reviews take place all the time. They are an essential part of the real world programming.
- There is no set way to do them.
- Review and Revise.

WHAT ARE FORMAL TECHNICAL REVIEWS?

Formal Technical Reviews (**FTR**) are formal examinations of software products to identify faults (i.e. departures from specifications and standards). They are a class of reviews that include:

- **System Definition Reviews**
- **Design Reviews**
- **Walkthroughs**
- **Inspections**
- **Test Readiness Reviews**
- **Functional & Physical Configuration Audits**
- **Formal Qualification Reviews**
- **Round-Robin Reviews.**
- **Other small group technical assessment of software.**

FORMAL TECHNICAL REVIEWS [Continued...]

These examinations can be applied to:

- **Requirements Specifications**
- **System Design**
- **Preliminary Design**
- **Detailed/Critical Design**
- **Program/Code**
- **Test Plans**
- **User Documentation**
- **Any other defined development product.**

OBJECTIVES OF FORMAL TECHNICAL REVIEWS

1 To uncover **errors** in function, logic or implementation of the software.

2 To verify that the software under review meets its requirements.

3 To assure that the software has been represented according to predefined standards.

4 To achieve software that is developed in a uniform manner.

5 To assure that members of the team have the same perception of system objectives.

6 To make projects more manageable.

7 Serve as Training grounds, enabling junior engineers to observe different approaches to software analysis, design and implementation.

8 Reviews also serve to communicate technical information widely in a project, supplementing formal written communication.

WHAT MAKES A REVIEW FORMAL?

1. A written report on the status of the product reviewed - available to everyone involved in the project, including management.

2. Active and open participation of everyone in the review group, following some written rules as to how such a review is to be conducted.

3. Full responsibility of all participants for the quality of the review.

DISTINCTIONS BETWEEN FORMAL AND INFORMAL REVIEWS

FORMAL REVIEWS

- Report to Management
- Results reported to all participants
- Each participant is responsible for the quality of the review
- The "Presumption of Guilt"

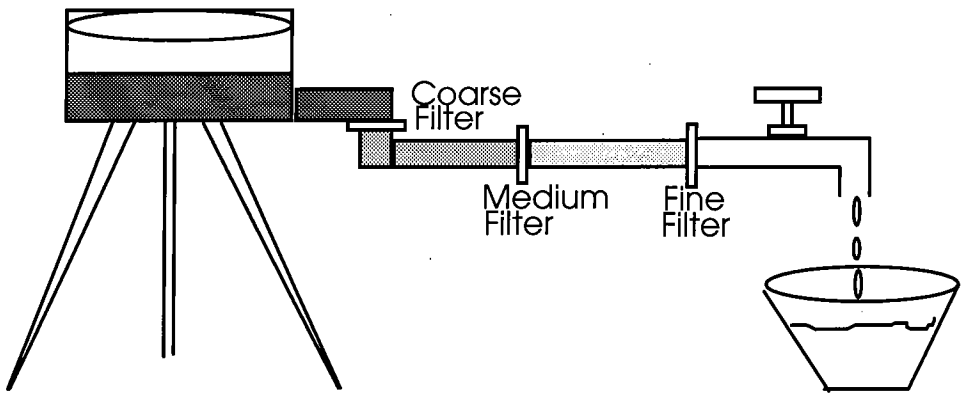
INFORMAL REVIEWS

- No Report to Management
- Results reported out to the author.
- Iterative Process.

WHY DO WE NEED REVIEWS?

1. Software reviews are "filters" for the software engineering process.

The water that comes out of the nozzle in your homes are initially full of impurities. If you don't filter the water, what you'll get from your tap is water full of impurities. Similarly, if you don't filter your software development process, what you'll get at the end is software full of bugs.



2. Technical work needs reviewing for the same reason that pencils need erasers: To err is human.

Because human beings are responsible for software development, there will be mistakes, and we need to make corrections when mistakes are made. The way to find mistakes is through formal technical reviews.

3. **Although people are good at catching some of their own errors, large classes of errors escape the originator more easily than they escape anyone else.**

This is the classical case of oversight. It's very common for people to overlook something not because they intentionally want to, but because in their minds, they think they did what they thought they wanted to do, even though that's not the case.

4. **To assure that members of the team have the same perception of system objectives.**

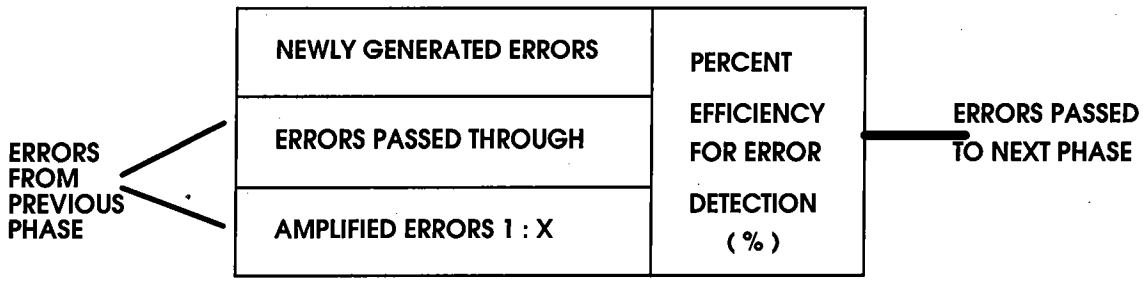
It's very important that everybody involved in a project (the user, designer, implementer, management, etc) understand what's required. You want everybody to perceive the system the same way. The way to do that is to conduct formal technical reviews.

DEFECT AMPLIFICATION MODEL

Unless defects are "**filtered**" (i.e., Found & Corrected) during the life cycle ...

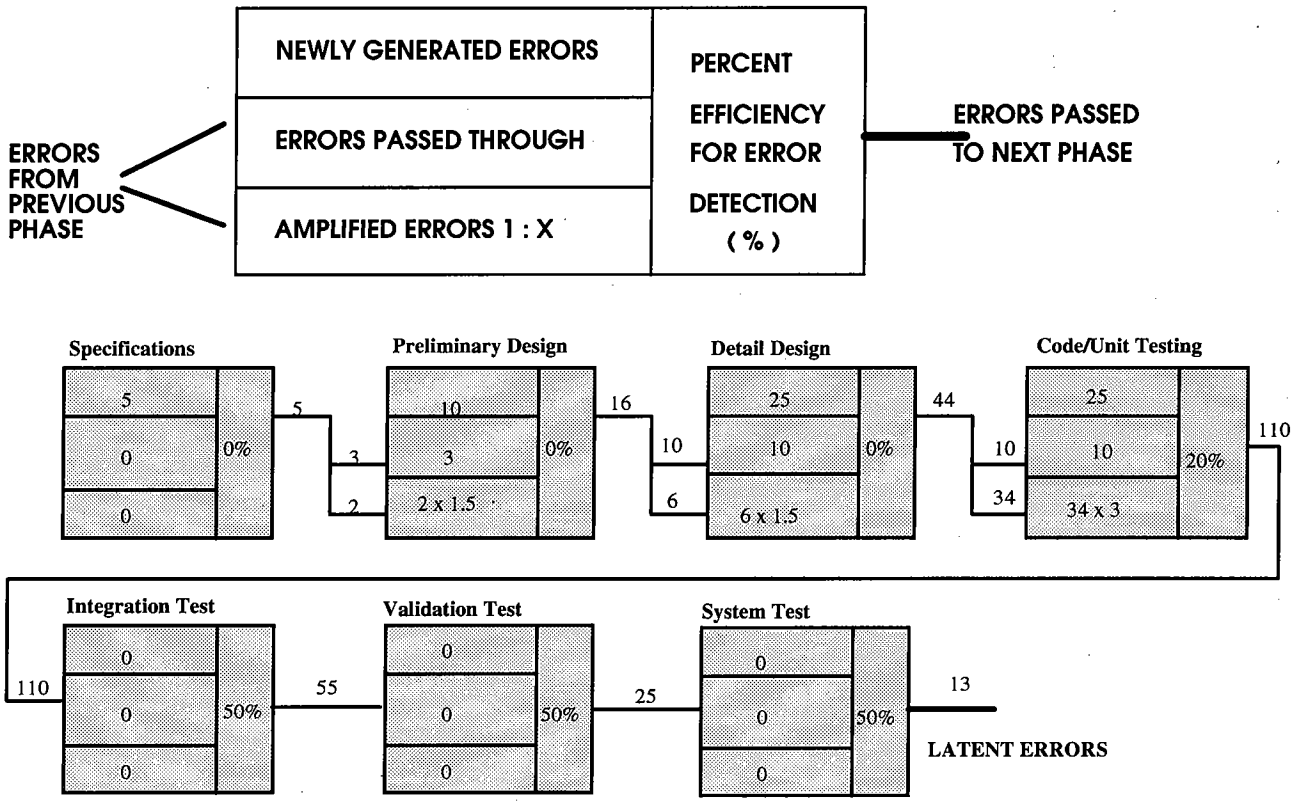
1. Some will pass undetected and be released to the field.

2. Others will be **amplified** during each step of the software development process.



NOTE: **X = amplification factor** (ranges from 1 to X)
where X depends on other factors such as the experience of the people involved, tools used, development environment, etc.

DEFECT AMPLIFICATION: WITH NO REVIEWS

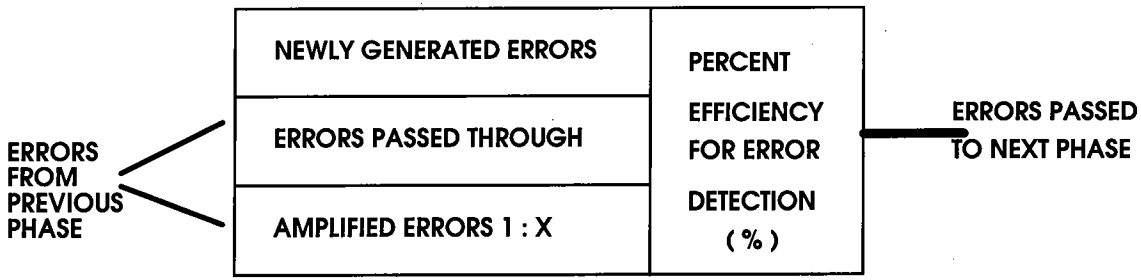


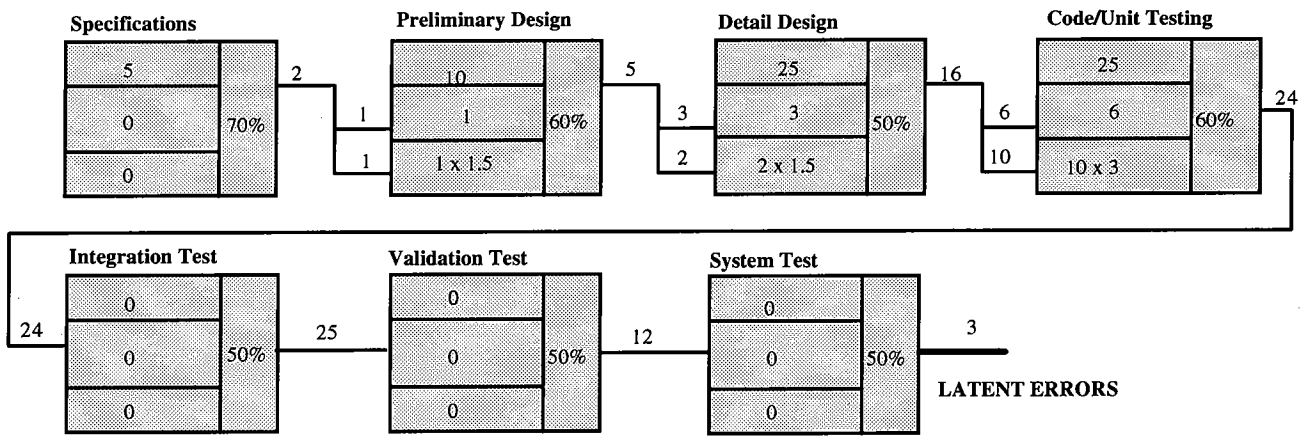
NOTE: **x = amplification factor**

ASSUMPTION:

Each test step uncovers and corrects 50% of all incoming errors without introducing any new errors. (Optimistic assumption)

DEFECT AMPLIFICATION: WITH REVIEWS





NOTE: x = amplification factor

ASSUMPTION:

Each test step uncovers and corrects 50% of all incoming errors without introducing any new errors. (Optimistic assumption)

RESULTS OF TWO ACTUAL EXPERIMENTS

Experiment 1: **Seeded 21 errors into a software module. The code was given to two groups of students - Group A and Group B. Group A performed only Unit Testing, while Group B conducted both Code Inspection and Unit Testing.**

(Conducted by Prof. Alan Kaminsky of the Software Development & Management Dept.,Rochester Institute of Technology, 1988)

RESULT:

	GROUP A	GROUP B
Total Errors Uncovered:	**11** (52.4%)	**19** (90.5%)

Experiment 2: **Errors were inserted into code. The code was given to two independent test groups - Group A and Group B. Group A did only Black-Box Testing, while Group B combined Black-Box Testing with Code Inspection.**

(Taken from Edward Showalter's presentation at the QAI Testing Conference in Orlando, Florida, December 1991.)

RESULT:

	GROUP A	GROUP B
Total Errors Uncovered:	**65%**	**85%**

COST IMPACT OF EARLY ERROR DETECTION

Assumptions:

1. An error uncovered during design will cost **1.5** monetary unit to correct.

Relative to this cost, the same error uncovered:

2. Just before testing begins will cost **6.5** monetary units.

3. During testing, **15** monetary units.

4. After release, **67** monetary units.

(Figures based on actual cost data collected for large software projects [IBM81].)
"Implementing Software Inspections," course notes, IBM Systems Sciences Institute, IBM Corp, 1981)
Roger S. Pressman, Software Engineering: A practitioner's Approach, 2nd ed., McGraw-Hill, Inc., 1987, pp.440-442)

DEVELOPMENT COST COMPARISON

	NO REVIEWS CONDUCTED			REVIEWS CONDUCTED		
Errors Found	**#**	**Unt-Cost**	**Total**	**#**	**Unt-Cost**	**Total**
During Design	0	1.5	0.0	25.5	1.5	38.25
Before Testing	27	6.5	175.5	37.0	6.5	240.50
During Testing	97	15	1455.0	21.0	15	315.00
After Release	13	67	871.0	3.0	67	201.00
			2501.5			794.75

HOW WE DID IT

Our Software Group: who we are

Our Software Environment

Getting started

Sustaining Momentum

Where we are today

OUR SOFTWARE GROUP: Who We Are

- 2 of 4 Software Groups
  Group 1 has 4 Software Engineers
  Group 2 has 12 Software Engineers

- 12 have formal education in computing

- Groups were just put together to work on this new project

- Responsibility is to develop real-time embedded software for the Consumer Imaging photofinishing community

OUR SOFTWARE ENVIRONMENT

IBM PC	PROTOTYPES **EVALUATION SYSTEMS** **ALGORITHMS**	DOS/C DEBUGGER
SUN SPARC STATIONS	PROTOTYPES **REAL-TIME, EMBEDDED SYSTEMS**	UNIX/C METRIC TOOLS

- SOME CROSS DEVELOPMENT

FORMAL TECHNICAL REVIEWS: Getting Started

- Agreed up front whether or not to do reviews
- Defined what reviews entail and which reviews to do
  (SQA Plan should outline what reviews to do, the entry and exit criteria for each review, what defects to track)
- Conducted a couple of "pilot" formal reviews
  (Development standards - Code standards; Software Plans - SQAP, SPP, SRS are good candidates)
- Assessed the "pilot" formal reviews process
  (Found out how the team felt about the process, What went wrong, what went well)
- Conducted a formal training on how to do formal reviews for the team
  (Allows members of the team to have the same perspective when they do reviews)
- Incorporated identified improvements to the formal reviews process
- Continued doing formal reviews
- Collect Metrics, assess process, and continuously improve on it

FORMAL TECHNICAL REVIEWS: Sustaining Momentum

- Ensure that the metrics being collected will be used for improving the process and nothing else, and make sure everybody understands that
- Listen to the participants and incorporate process improvement suggestions by the team into the process
- Share the results of each review session with the participants
  (Reviewers feel good about themselves when they see that they indeed identified errors)

GUIDELINES FOR REVIEWS

Guidelines for the conduct of Formal Technical Reviews should be established in advance, distributed to all reviewers, agreed upon, and then followed.

1. Review the product, not the producer.

FTR involves people and egos. Errors should be pointed out gently; the tone of the meeting should be loose and constructive; the intent is not to belittle or embarrass.

2. Set an agenda and maintain it.

A malady of any meeting is drift. An FTR must be kept on track and on schedule.

3. Limit debate and rebuttal.

When an issue is raised by a reviewer, if there is no universal agreement on its impact, record the issue for further discussion off-line, rather than spend time debating the question.

4. Don't attempt to solve the problem.

A review is not a problem-solving session. Problem solving should be postponed until after the review meeting.

5. Take written notes.

Sometimes a good idea for the recorder to make notes on a wall board so that wording and prioritization can be assessed by other reviewers as the information is recorded.

6. Limit the number of participants.

Two heads are better than one, but 14 are not necessarily better than 4.

7. Insist upon advance preparation.

All review members must prepare in advance. Written comments should be solicited by the review leader.

8. Develop a checklist for each product that is likely to be reviewed.

A checklist helps the review leader to structure the FTR meeting, and helps each reviewer to focus on important issues.

9. Allocate resources and time schedule for FTRs.

To be effective, FTRs should be scheduled as a task during the software engineering process. In addition, time should be scheduled for the inevitable modifications that will occur as the result of an FTR.

10. Review your early reviews.

Debriefing can be beneficial in uncovering problems with the review process itself. The very first product to be reviewed might be the review guidelines themselves and your development standards.

11. Restrict a Design Review to Reviewing One Design.

Don't use a design review to compare two or more designs, but use two or more design reviews to compare two or more designs. When you try to do two designs at once, the review may turn into a yelling contest for the advocates of the various alternatives.

12. Conduct meaningful training for all reviewers.

To be effective, all review participants should receive some formal training. (Freedman & Weinberg [FRE82] estimate a one-month learning curve for every 20 people who are to participate effectively in reviews).


Don't Ever Give Up!