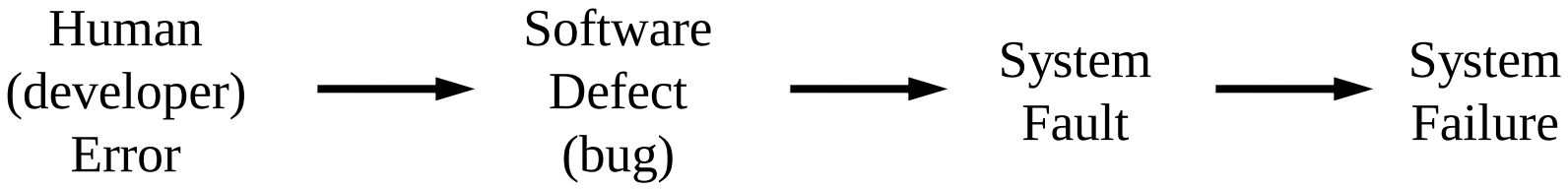
Defect Prevention and Removal

Objectives

- Provide basic concepts about defects and defect-oriented practices
- Practices dealing with defects
  - Setting defect removal targets:
    - Cost effectiveness of defect removal
    - Matching to customer & business needs and preferences
  - Performing defect prevention, detection, and removal
    - Techniques/approaches/practices overview
- Introduce some basic measurements and metrics for tracking defects and defect practice performance
  - Defect measurements and classification
  - Measurement source: Inspections, test reports, bug reports
  - Defect density
  - Phase Containment Effectiveness
  - Cost of Quality & Cost of Poor Quality
  - Tracking of bug fixing and fixing effectiveness

Terminology of Causality: Anomalies

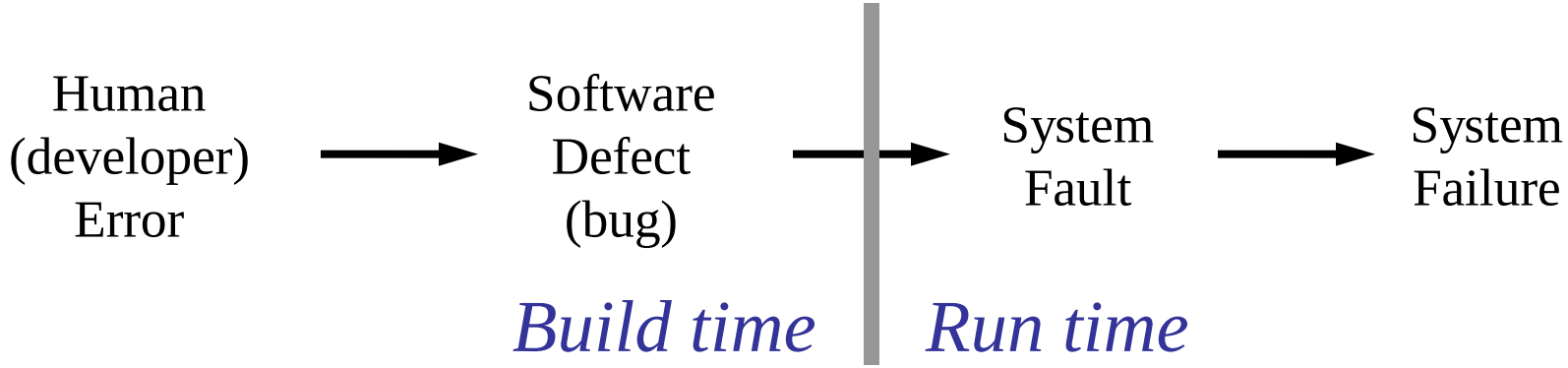


[IEEE Std 1044-1993—IEEE Standard for Classification of Software Anomalies]

"… use of the word *anomaly* is preferred over the words *error*, *fault*, *failure*, *incident*, *flaw*, *problem*, *gripe*, *glitch*, *defect*, or *bug* throughout this standard because it conveys a more neutral connotation."

anomaly: Any condition that deviates from expectations based on requirements specifications, design documents, user documents, standards, etc., or from someone's perceptions or experiences. Anomalies may be found during, but not limited to, the review, test, analysis, compilation, or use of software products or applicable documentation.

Avoid Build-Time Defects and Run-Time Faults



- Software engineering processes attempt to remove human errors
- Software reviews and inspections attempt to remove software defects before they appear at run-time
- Software testing attempts to cause software defects to manifest themselves as observable faults (and failures) at run-time
- A fault-tolerant system may stop run-time faults from becoming run-time failures
- Failure containment attempts to limit the impact of a system failure

Quality Views

- People expect the software they use and rely on to:
  - Do the right things
  - Do the things right
- Correctness focus: Correct functionality under all conditions
- Developer's perspective: No defects in functionality
- User's perspective: No failures of functionality

- But there is more to quality that correct functionality

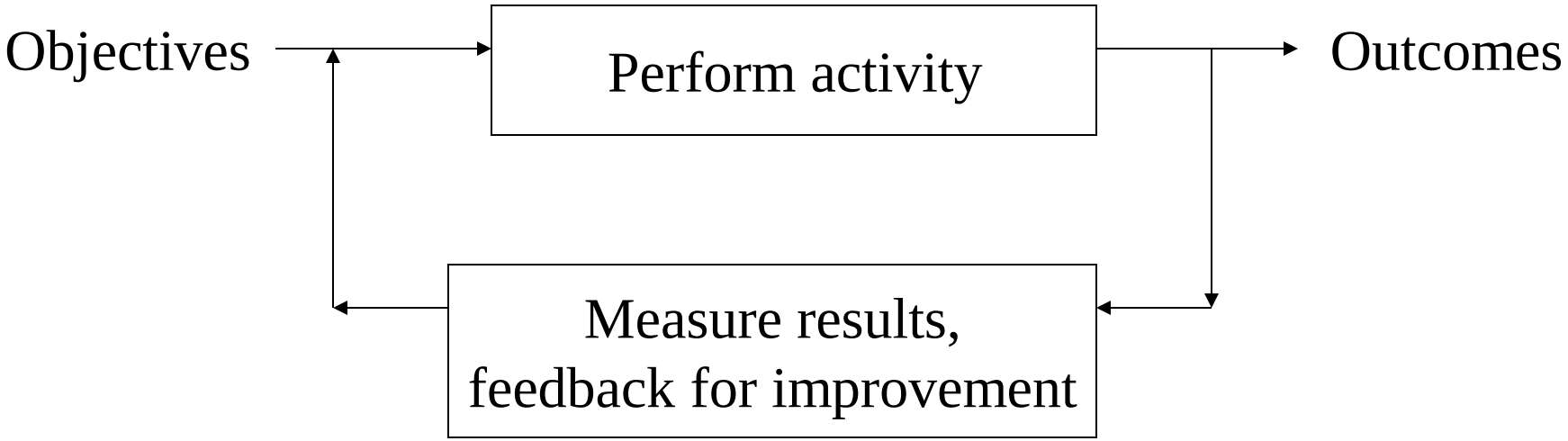
Quality Views and Attributes

View	Attribute	
	Correctness	Other
Customer (external)	Failures: reliability safety etc.	Maintainability Readability Portability Performance Installability Usability, etc.
Developer (internal)	Faults: count distr class etc.	Design Size Change Complexity presentation control data, etc.

Engineering Practices on Defects

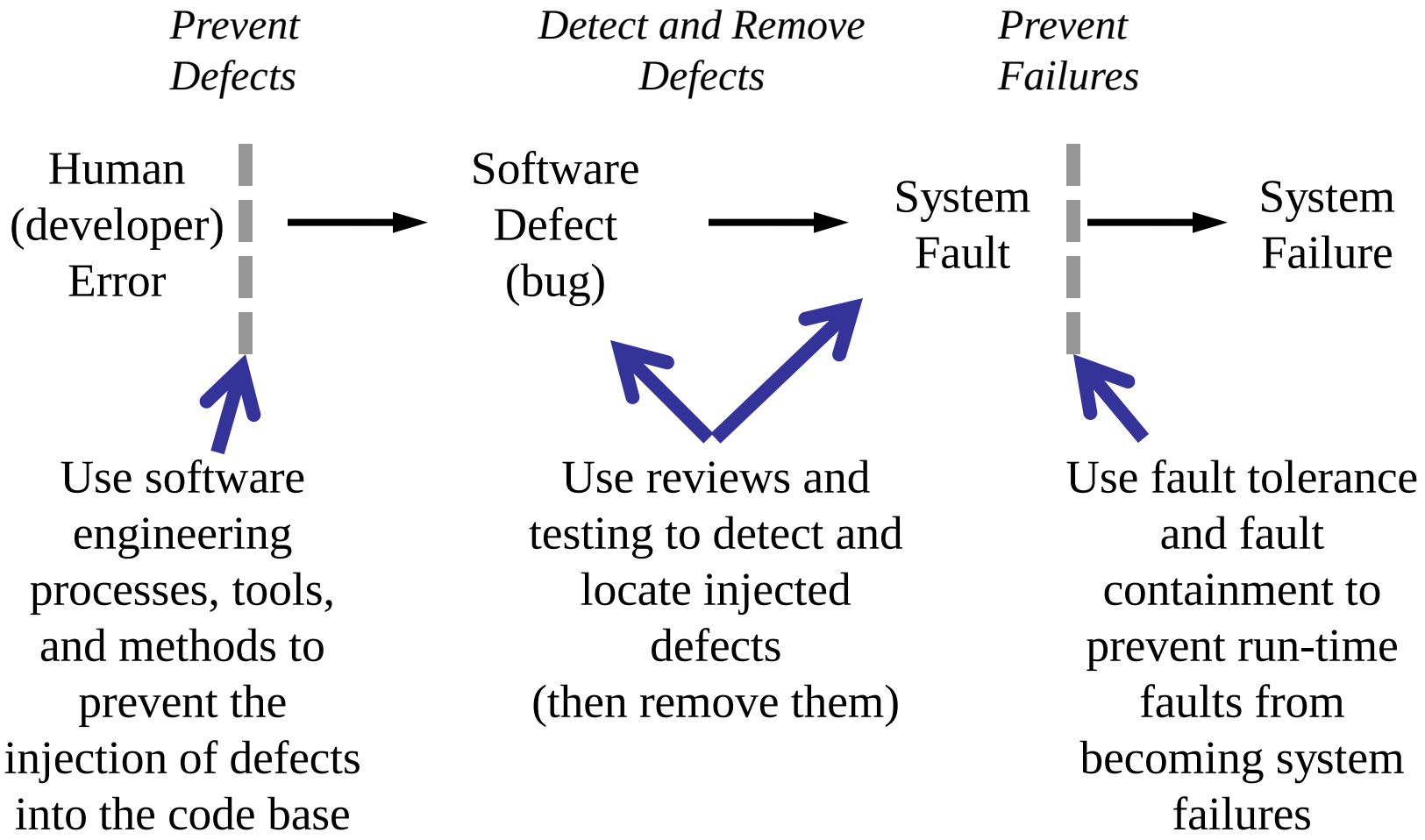
Underlying Quality Engineering Model

- Optimizing results using feedback:



In the next few weeks, we take different SE areas – Defect Prevention and Removal, Product Quality, Customer Satisfaction, Project Management – and study the quality engineering objectives, practices and metrics for each area.

Prevent, Detect, and Remove Defects Prevent Failures



Defect Removal Objectives

Note: "Defect Removal" is often used as shorthand for defect prevention, detection, and removal

- Low defect density in product
  - Different density targets depending on defect severity level
  - Actual targets based on nature of software:
    - Impact of defects, expectations of customer
      - (Will discuss in more detail under reliability)
    - Often the idea of "setting a defect rate goal" is not discussable
    - What about the goal of "no known defects"?
      - Is shipping with known defects acceptable?
- Cost-effective defect removal:
  - Quantitative understanding of which approaches most cost-effective
  - Quantitative understanding of how much effort is worthwhile

Defect Removal Practices 1

(Practices grouped by increasing sophistication of approach)

- Informal defect removal:
  - Informal discussion and review of requirements with customer
  - Sporadic testing prior to release
  - Informal discussions and reviews within team
  - Informal defect reporting and fixing
- Informal but strong quality focus:
  - Extensive testing
  - Creating test cases, writing test code
  - Feature-based testing
  - Need-based inspections and reviews
    - "This code seems to have problems, let us improve it"

Defect Removal Practices 2

- Test strategy and test planning:
  - Informal attempts at coverage
  - Systematic identification of test cases
- Tracking of detected problems to closure for both tests and reviews
- Systematic customer and developer reviews of generated documents
- Tracking of defect/failure reports from customers
- Possibly some use of test automation
  - Test harnesses that systematically run the software through a series of tests
- Practices that prevent some kinds of defects
  - Training, configuration management, prototyping

Defect Removal Practices 3

- Formal peer reviews of code and documents
- Tracking of review and test results data
- Use of coverage analysis and test generation tools
- Tracking of data on defect fixing rates
- Use of graphs showing defect rates for informal diagnosis and improvement
- Improved defect prevention using checklists, templates, formal processes

Defect Removal Practices 4

- Use of metrics to:
  - Analyze effectiveness of defect removal
  - Identify problem modules (with high defect densities)
  - Identify areas where practices need strengthening
  - Identify and address problems “in-process” – during development
  - Set defect density / defect rate objectives
    - Usually “baselines” – current capability level
  - Guide release (only when defect detection rates fall below threshold)
  - Plan test effort and number of test cases
  - Optimize quality efforts
- Consistent use of test automation
- Use of formal methods for defect avoidance

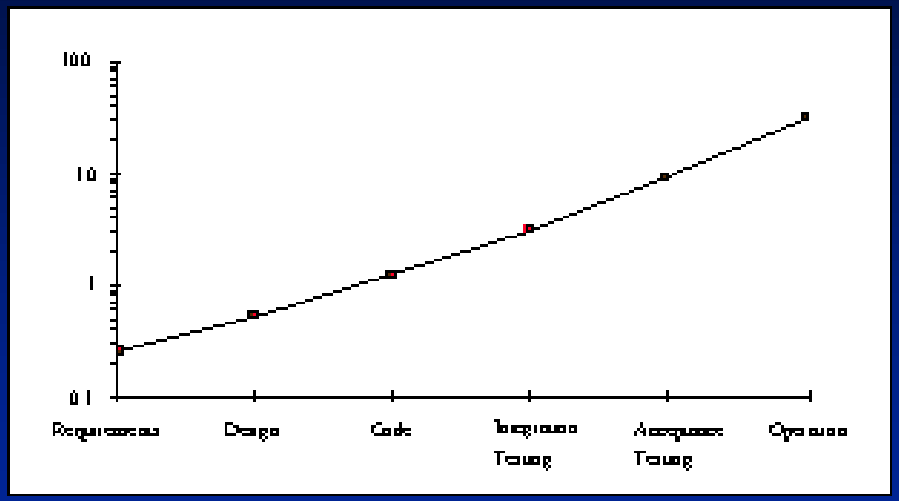
Defect Removal Practices 5

- Continuous improvement cycle
  - Pareto analysis to discover common sources of problems
  - Causal analysis to identify roots of frequent problems
  - Use defect elimination tools to prevent these problems
  - Repeat!

Value of Early Defect Detection

Defect Removal Costs Increase Radically During Software Life Cycle

Relative Cost of Defect Removal



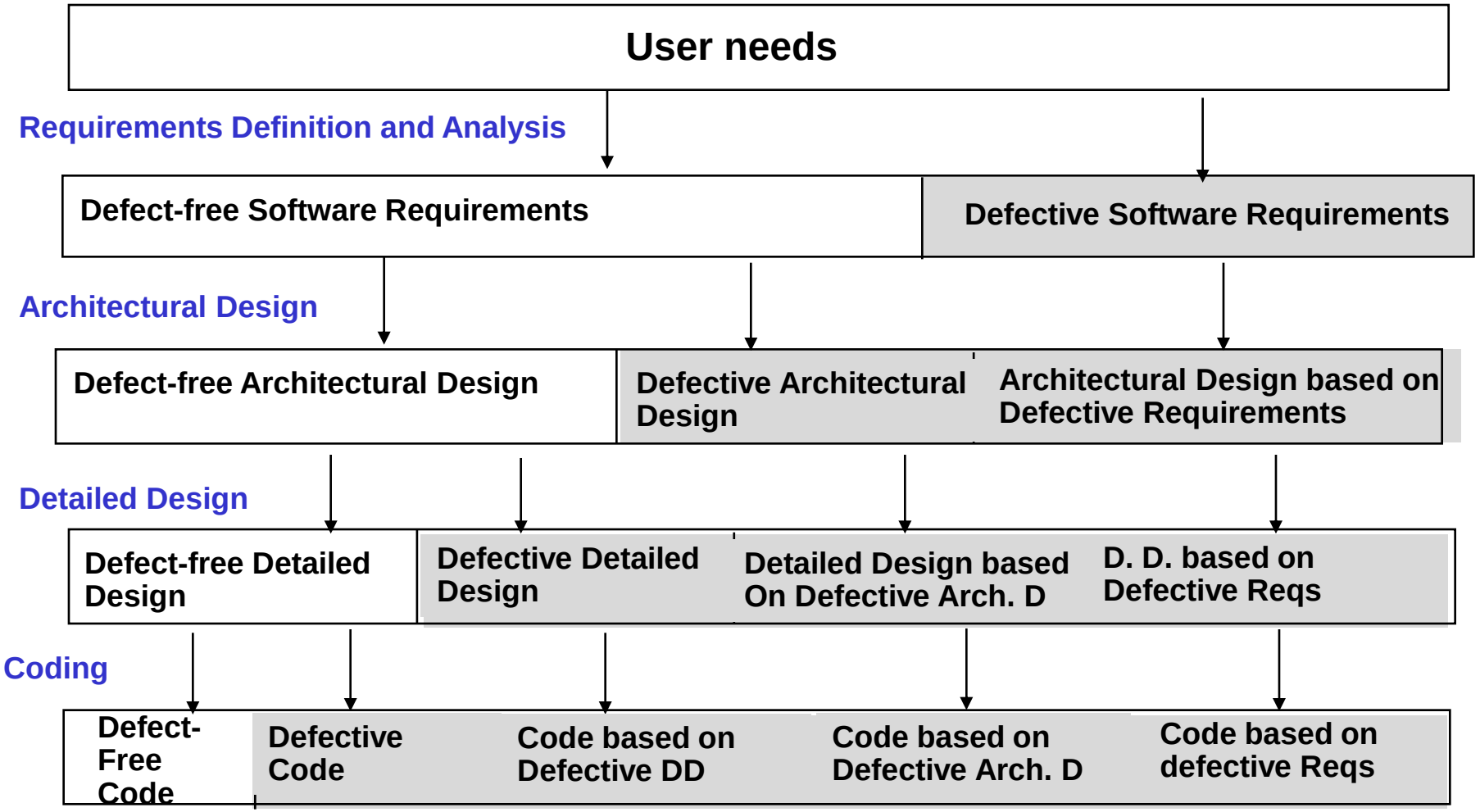
The Lesson: Find Defects Early!

 Rev. 1998-4 Slide 16

Note that y-axis scale is logarithmic – actual increase is exponential

From http://www.sdtcorp.com/overview/inspections/sld016.htm

Defect Injection and Propagation



How to Detect Defects Early?

- Inspections and reviews
- Prototyping, extensive customer interaction
  - Note that agile development emphasizes these
- Use of analysis techniques:
  - Requirements analysis for completeness and consistency
  - Design analysis, such as sequence diagrams to analyze functional correctness, quality attribute analysis
  - Formal specification and analysis of requirements
- Methodologies that increase early lifecycle effort & depth:
  - O-O development increases design effort & detail
  - Test-driven development increases understanding of relationships between design and requirements
  - Traceability analysis
- Incremental development to expose defects in early operation

Need for Multi-Stage Approaches

- (Just an illustrative example)
- One phase of defect removal, such as testing:
  - Assume 95% efficiency (called PCE: *phase containment effectiveness*)
  - Input 1000 defects – output 50 defects
- Six phases of defect removal, such as Requirements, Design, Implementation, Unit Test, Integration Test, System Test
  - Assume 100, 300, 600 bugs introduced in Requirements, Design, Implementation, respectively
  - Even with much less efficiency, we get better results
  - 10% improvement in PCE produces 3x better results
- Similar concept for incremental development: Increment containment effectiveness

Phase	Req	Des	Impl	UT	IT	ST
60% eff: defects at entry	100	340	736	295	118	47
60% eff: defects at exit	40	136	295	118	47	19
70% eff: defects at entry	100	330	699	210	63	19
70% eff: defects at exit	30	99	210	63	19	6

Defect Data

Sources of Defect Data

- Inspection / review reports contain
  - Module
  - Defect type (see next slide on defect classification).
  - Defect severity
  - Phase of detection
  - Effort data: review prep, review meeting, effort to fix problems
  - Number of lines of code reviewed
- Similar data gathered from testing
- User defect/failure reports
  - Manual screening to reject duplicates, non-problems
  - Similar classification and effort data

Defect Classification

- Many organizations have their own defect classification system:
  - For example, defect type: Logic, requirements, design, testing, configuration mgmt
  - May classify in more detail: initialization, loop bounds, module interface, missed functionality, etc.
    - Helps in Pareto analysis for continuous improvement
    - More effort, less reliable: errors in classification, subjectivity
  - Did defect originate from previous fix?
- There exists a methodology called “Orthogonal Defect Classification” (ODC)
- For more details, see IEEE Standard for Classification of Software Anomalies (IEEE Std 1044-1993)

Processing of Defect Data

- Compute phase containment effectiveness based on:
  - Number of defects found in that phase
  - Number of defects from that phase or earlier found subsequently
- Similarly, compute test effectiveness
- Fixing effectiveness
  - Did the fix actually remove the defect?
  - Did the fix inject new defects?
- Overall defect density
- Density per module, phase, increment
- Review rates: number of lines / hour
- Cost of quality: Total effort spent on quality activities
- Cost of poor quality: Total effort spent on fixes

Limitations in Defect Data

- Small sample sizes:
  - Smaller projects often have < 100 defects
  - Classifying by type, phase etc. further reduces the "population" from statistical perspective
  - PCE in particular has sample size problems
  - Organizational numbers often more meaningful
- PCE reduces as more bugs found!
  - Hard to use as in-process metric
- Subjectivity of classification
- Developers may suppress defect data to look good
  - Fundamental rule: "Never use metrics to evaluate people!"

Conclusion

- Provided some basic concepts and practices
  - Anomalies: Error → Defect → Fault → Failure
  - Prevent defects and failures
  - Detect and remove defects
  - Early defect detection and increment/phase containment effectiveness
- Numerous ways to classify and analyze defect data