



# Using Defect Patterns to Uncover Opportunities for Improvement

ASM 99 Metrics Conference  
February 1999

**Paul Matthews**  
**Bellcore Applied Research**  
**pmatthew@bellcore.com**  
**973-829-4766**

# Outline

---

- Context of ODC Use in Bellcore
- Basics of Defect Pattern Analysis
- Case Study Example
- Potential Pitfalls
- Future Directions

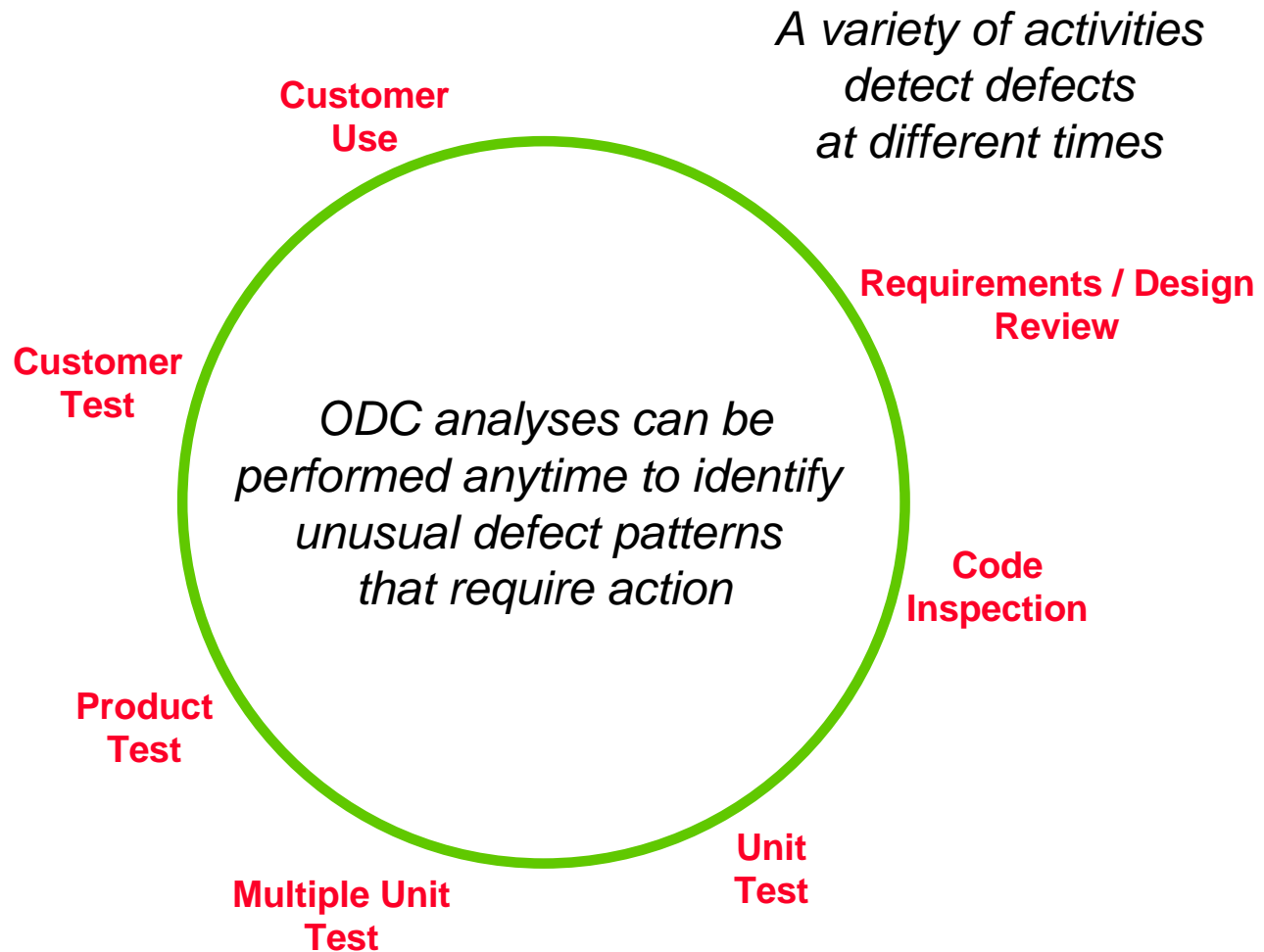
# ODC at Bellcore

---

- ODC involves classifying software defects using a predefined set of categories
- General goal is to enable software product teams to improve engineering practices by discovering, investigating, explaining, and correcting unusual patterns of defects

# Integrated Processes

---



# ODC Categories

---

- When defect is detected
  - Lifecycle Phase (when detected)
  - Trigger (how detected)
  - Impact (effect on customer)
  - Severity (priority to fix)
- When fix is known
  - Defect Modifier (missing or incorrect)
  - Defect Type (at program code level)
  - Defect Source (code history)
  - Defect Domain (generic subsystem)
  - Fault Origin (requirements, design, or implementation)

# Episodes

---

- An “episode” is a distinct work effort delimited in time, often involving different people
- Values used to define episodes
  - Product / Subsystem (which body of software)
  - Release (which package)
- Episodes can be combined for analysis purposes

# Defect Pattern Defined

---


- By “pattern” we mean the proportion of defects falling into each category value, or combination of category values
  - Total number of defects is another issue
- Patterns are typically represented by N-way tables, bar charts and other graphics
- Patterns are calculated per episode

# Example Defect Pattern

**N = 132**

		Lifecycle Phase			
		Code Inspection	Unit Testing	Multiple Unit Testing	Product Testing
Defect Modifier	Missing	2%	3%	11%	20%
	Incorrect	16%	19%	17%	12%

*Unusual pattern elements are highlighted*

 Higher than expected

 Lower than expected

# Example 2-Way Classification Pattern (cont.)

---

- Verbal description
  - “The percentage of Missing defects found in Code Inspection and in Unit Testing is unusually low, and the percentage found in Product Test is unusually high.”
- Hints for investigation
  - “The procedures used for Code Inspection and Unit Test may have been narrowly focused on the existing code, and did not compare the code with the requirements and design documents.”

# Category Value Restrictions

---

- May be needed to compare incomplete data
  - E.g., one episode includes all the defects found in Unit Test and another episode does not
- Some category values are not always meaningful
  - E.g., an episode has a high proportion of “Unknown” or “Other”
- Drill down may be simpler than N-way classification
  - E.g., special interest in Severity 2 field defects

# Imperfect Data

---

- Analysis is relatively easy when there are lots of good quality data for comparisons
  - Even a small number of defects can be compared successfully against a high definition background
- But reality can have complications
  - Data may not (yet) be in the repository
  - Products differ and some are not as good comparisons
  - Defects may not be as reliably classified for some lifecycle phases and severity levels, depending on the project
- Looking at meaningful subsets is a basic tactic
  - Select data by episode and category values

# Early Bootstrapping Trials

---

- Adapted ODC category definitions to our business
- Classified software defects retrospectively
  - E.g., for each major release
- Collected data using a spreadsheet application
- Analysis depended on inspecting tables/charts and recognizing unusual patterns
- Winning points
  - Classifying defects and getting the big picture before diving into detailed records saved a lot of time
  - Classifications provided a basis for more understandable stories, explanations, and proposals for action

# Scaling Up Issues

---

- Retrospective classification is too slow
  - Want data to manage processes in real time
  - Easier to classify defects as they are recorded
- Number of patterns is large for simple visual inspection
  - E.g., 9 categories implies 36 2-way and 94 3-way combinations
  - Mechanical assistance needed to avoid missing patterns
- Data are noisy
  - Patterns are sometimes seen in random variation
  - Statistical filter needed to reduce noise
- Analysis and follow up resources are limited
  - Limited time, need to get to the point quickly
  - Concentrate on a few problem areas

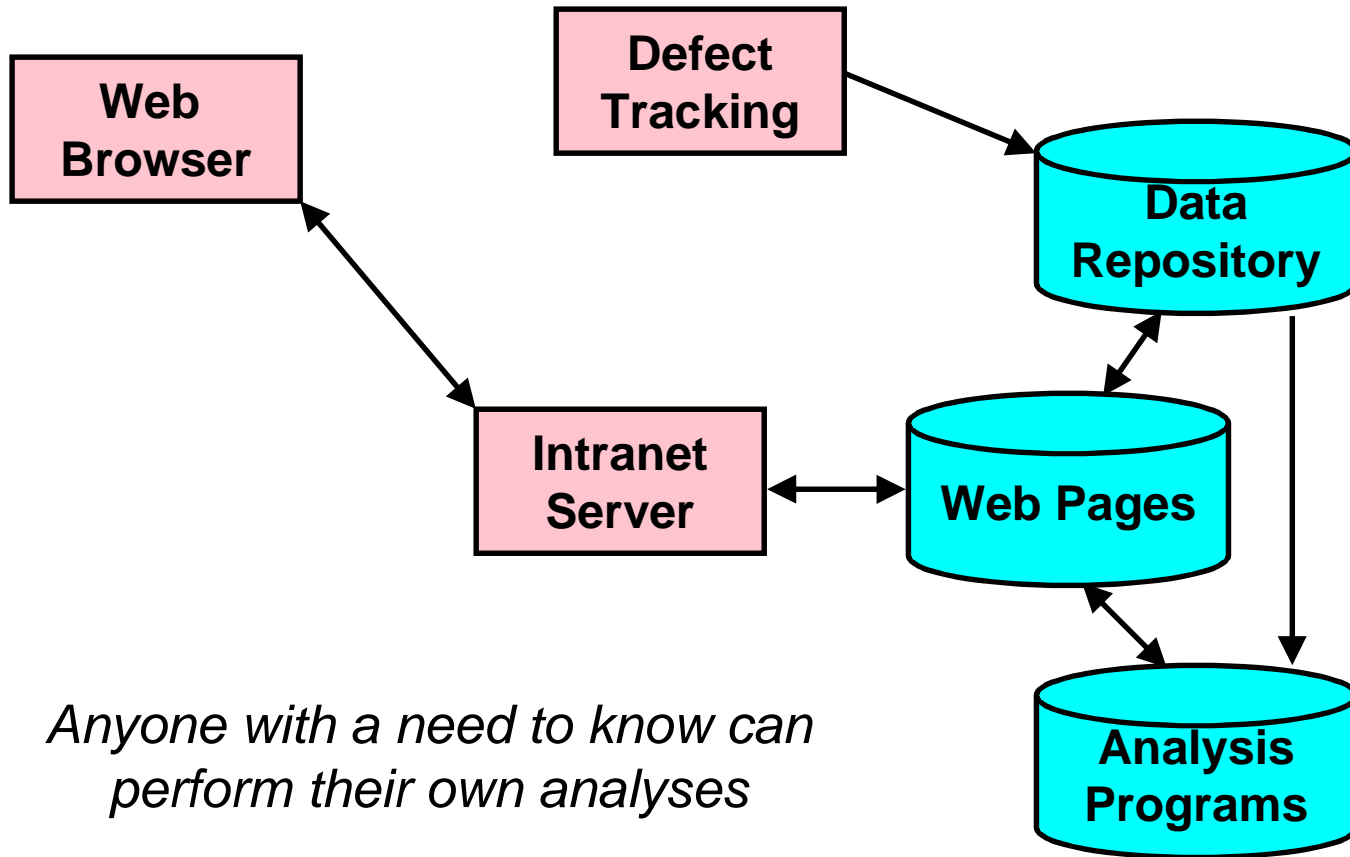
# Scaling Up Approaches

---

- Classification done in defect tracking systems
  - Entails widespread awareness
- Web based access to corporate data repository
  - Enables data sharing and comparisons
- Statistical software to highlight salient data
  - Simplifies what the user needs to notice
- Integrated tools package
  - “Efficient Defect Analyzer” (EDA)

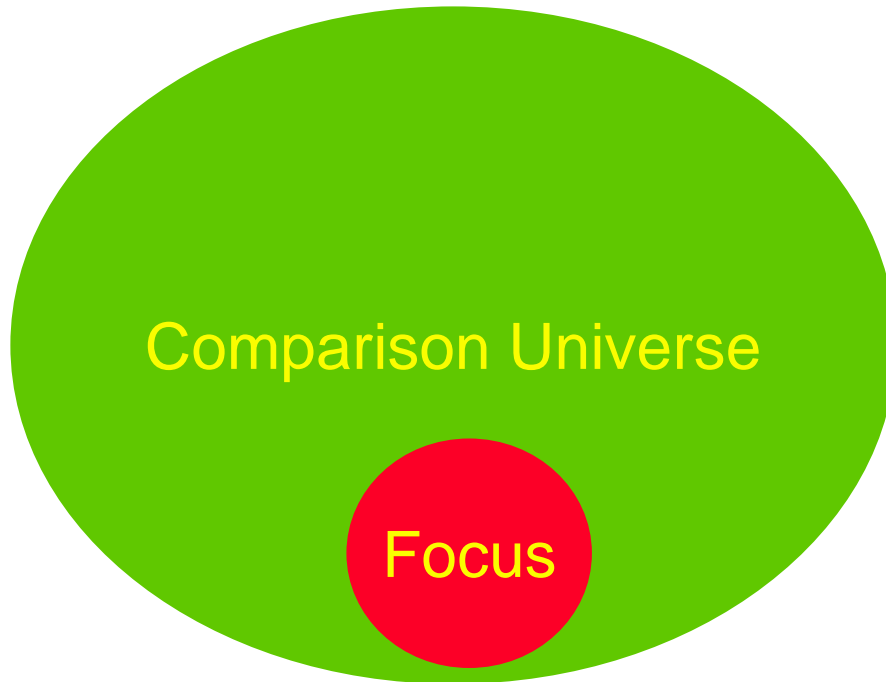
# Web Based Infrastructure

---



# EDA Comparative Analysis Concept

---



*User selects focus episodes within a universe of episodes for comparison*

- Simultaneous test for overall differences
- Individual tests for high/low elements

# Overview of Analysis Steps

---

- Select episodes for analysis
- Select episodes for comparison
- Specify any category value restrictions
- Note any unusual pattern elements
- Consider additional comparisons
- Write the story from the notes
- Investigate to confirm and elaborate the story
- Propose any actions that should be taken

# Case Study Example Scenario

---

- Defect analysis team for Product A
  - 1 tester, 2 developers, 1 metrics consultant
- Analyzing field defects was the main concern
- Focus episodes were
  - Product A Releases 2 and 3
- Available comparison episodes were
  - Product B Release 3
  - Product C Release 3
- Used EDA web based tools for ODC analysis

# ***Case Study Example***

## **Combining Product A Releases 2 and 3**

---

- Performed EDA analysis comparing releases 2 and 3
  - Found no significant differences (i.e., no unusual pattern elements were highlighted)
- Decided to combine releases 2 and 3 as one focus episode group
  - Team thought that releases 2 and 3 were similar
    - same kinds of functionality
    - same technology platform and tools
    - same software engineering processes
    - same staff turnover
  - Combining gives greater power for other comparisons

# ***Case Study Example***

## **Comparing Product A With B, Sev 2**

---

- Data found to be incomplete
  - Product B had only severe field defects
  - So, only severe field defects were compared
    - 10 for Product A, 38 for Product B
- Unusual defect pattern elements (click [here](#))
  - *Impact*: Low percentage of Capability problems
  - *Impact x Source*: High percentage of Reliability problems concentrated in Old Functionality
  - *Trigger x Domain*: High percentage of Start/Restart problems in Server Processes

# Case Study Example

## Comparing Product A with C, Sev 2

---

- Product C was larger, with more components
  - 134 defect records provide higher definition background
- Unusual defect pattern elements (click [here](#))
  - *Trigger*: High percentage of Workload Stress problems
  - *Defect Type x Trigger*: High percentage of Algorithm problems under Workload Stress
  - *Defect Type x Impact*: High percentage of Algorithm problems have Reliability impact
  - *Defect Source*: High percentage of Old Functionality problems
  - *Defect Source x Impact*: High percentage of Old Functionality problems have Reliability impact

# Case Study Example

## Comparing Product A with C, Sev 3

---

- Less severe field defects
  - 175 defect records
- Unusual defect patterns (click [here](#))
  - *Domain*: High percentage of GUI problems
  - *Domain x Impact*: High percentage of GUI problems with Usability impact
  - *Domain x Fault Origin*: High percentage of GUI problems due to Implementation
  - *Fault Origin*: High percentage of Requirements problems
  - *Fault Origin x Domain*: High percentage of Requirements problems for Server Processes
  - *Defect Type x Trigger*: High percentage of Algorithm problems found under Workload Stress

# ***Case Study Example***

## **Comparing Product A By Itself, Sev 2**

---

- Compared null hypotheses
  - All category values are equally frequent
    - Uniform percentages
  - Categories are independent
    - Marginal percentages predict joint percentages
- Severity 2 only
  - 10 defect records
- No unusual defect patterns were found
  - Not as good contrast as real comparisons

# ***Case Study Example***

## **Comparing Product A By Itself, Sev 2 & 3**

---

- Both severity 2 and 3
  - 81 defect records
- Comparison with null hypotheses highlights pattern elements (click [here](#)), but many have little meaning because they are fairly common across episodes
- Distinguishing the more meaningful pattern elements is difficult without real comparisons

# *Case Study Example*

## Putting the Story Together

---

- “There were a relatively high number of reliability problems concentrated in older code.”
- “Workload stress revealed a relatively high number of algorithm errors in older code that had severe reliability impacts.”
- “A relatively high number of server problems were found during system restarts.”
- “There were a relatively high number of GUI problems. However, these problems only affected usability and were not severe. A relatively high number of the GUI problems were due to implementation errors.”

# ***Case Study Example***

## **Investigating the Story**

---

- Trace back from highlighted cells to detailed defect records
  - Check classifications
  - Elaborate the story
- Tell the story to other project members
  - Check credibility
  - Gather additional facts and ideas

# Case Study Example

## Explanations

---

- “An optimization algorithm, programmed in Albert release 1, failed in cases where high arrival rates caused a queue to grow beyond an expected limit. When a failure occurred, a database record could be stranded in an inconsistent state, causing a restart to fail.”
- “New hire GUI programmers did not fully understand how to use the programming environment to implement the GUI design, and failed to handle some events properly. Also, as the design evolved, some of the online messages to the user got out of sync with the functionality.”

# ***Case Study Example***

## **Proposed Actions**

---

- "Provide more algorithm details in design documents, including assumptions about the runtime environment and possible limitations. In both design reviews and code inspections, reviewers should require information about the performance of algorithms under extreme conditions."
- "Exception handling code should be written to take over when expected limits are exceeded. Product testing should include high stress cases, and verify that exception handling code has been executed successfully."

# ***Case Study Example***

## **Proposed Actions (cont.)**

---

- "Create a list of all information messages that can be sent to a user. Testing should include cases that invoke these messages."
- "Senior programmers should conduct monthly seminars on GUI design and programming; attendance should be mandatory for everyone involved in developing GUIs. A senior programmer with design knowledge should be involved in every GUI code inspection."

# ***Case Study Example***

## **How Good Was This Analysis?**

---

- Analysis was completed in one day, start to finish
  - Traditional root cause analysis would have consumed more than 5 times the resources
- Analysis and proposals were credible
  - Story made sense to project members
  - Quantitative data helped convince managers
- Having more complete data would have helped
  - E.g., other lifecycle phases, all severity levels
- More discussion with colleagues who support products B and C would have strengthened proposals
  - Limited by time pressure, reluctance to reveal problems

# Analysis Pitfall #1

## Already Knowing the Answers

Write conclusions  
and seal in envelope

*Naïve data analyses often  
come out this way*

Collect and analyze data

Open the envelope and  
present to management  
(with supporting data)

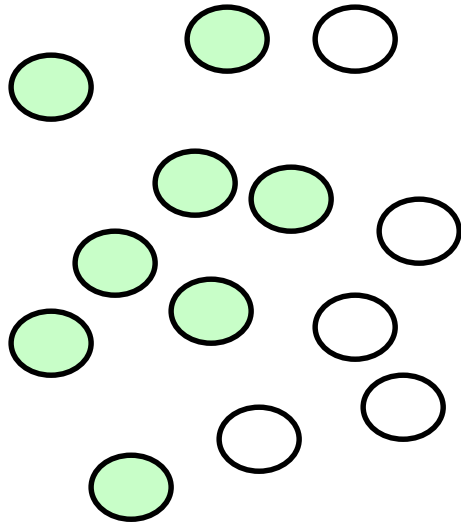
*What to do about it:*

- Challenge project team to explain specific numbers
- Review together with corporate experts

# *Analysis Pitfall #2*

## Chronic Incomplete Data

---



*Some defects may never appear in the data, which can bias analysis*

*What to do about it:*

- Uniform rules and thresholds for recording defects
- Formally open and close reporting windows

# Analysis Pitfall #3

## Improper or Inconsistent Classification

---

### Frequent Weird Values



*Likely to show up as unusual patterns that are hard to explain*

*What to do about it:*

- Training using examples drawn from own project
- Peer review within project
- Infrastructure capability to trace back to source

# Analysis Pitfall #4

## Pounding Down Nails

---



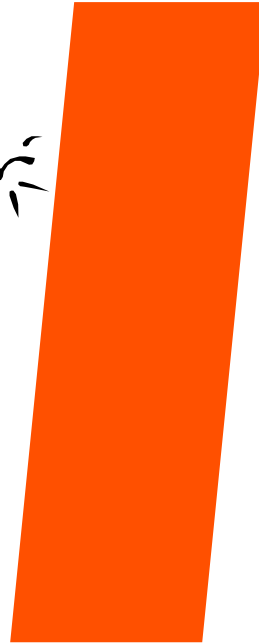
*Relatively higher frequencies  
are not necessarily bad*

*What to do about it:*

- Emphasize pattern as a whole
- Require comparisons with other products

# Analysis Pitfall #5

## Isolation



*A project can learn only so much  
from its own experience*



*What to do about it:*

- Require comparisons with other products
- Encourage discussions between projects

# Things We Have Learned

---

- Classification changes the way people think
  - More power to generalize and find simple solutions
  - More power to distinguish cases that require different treatments
- Trying to explain real numbers forces a fresh look
  - Preconceptions are often insufficient to explain simple facts
  - Draws out stories that would rarely be mentioned otherwise, and new ideas about what's happening
- Automated analysis infrastructure is important
  - Helps analysts get focused quickly and use limited resources productively
  - Expert consultants are not always available
- Data quality requires vigilance
  - Widely distributed responsibility

# Future Directions

---

- Classifying defects before software exists
  - Negotiations, contracts, schedules, requirements, specifications, designs, etc.
  - Implications for project management
- Analysis software that writes the story and makes its own recommendations
  - Users would still need to do some reality checks and provide feedback to the software
- Advanced visualization that melds virtual reality and statistical highlighting
  - E.g., manipulation of 3D models in a web browser