ORTHOGONAL DEFECT CLASSIFICATION

Billg Fall 1997 Retreat: Improving the Software Development Processes at Microsoft

Michael Herman

Senior Consultant

Microsoft Canada Co.
mherman@microsoft.com

The Right Tests at the Right Time

- “Testing Readiness”
  - *Knowing when a feature is ready to test*
  - *Knowing he right tests to run at the right time*

- Can ODC concepts be applied to code artifacts to determine the completeness, stability and testability of a code base?

Outline

- Framework for understanding ODC
- “Traditional” Orthogonal Defect Classification
- How ODC’s concepts can applied to Testing Readiness
- Next steps

General Framework

- Select the object you want to study
  - *the thing you want to learn more about*
  - *examples: the way a team develops s/w, code base*
- Identify artifacts of the object under study
  - *each artifact has certain attributes (and attribute values associated with it)*
  - *examples: defects, source code change events, resp.*
- From a database of attribute values, extract useful knowledge about the object of study

General Framework

- The Method
  1. *Data collection and classification*
  2. *Clean-up and pre-processing of the data*
  3. *Data mining* *(to enumerate patterns)*
  4. *Interpretation* *(leading to knowledge discovery)*
  5. *Actions* *(based on the extracted knowledge)*
- Acts as a <u>feedback loop</u>

What is Traditional ODC
(and how it might be used at MS?)

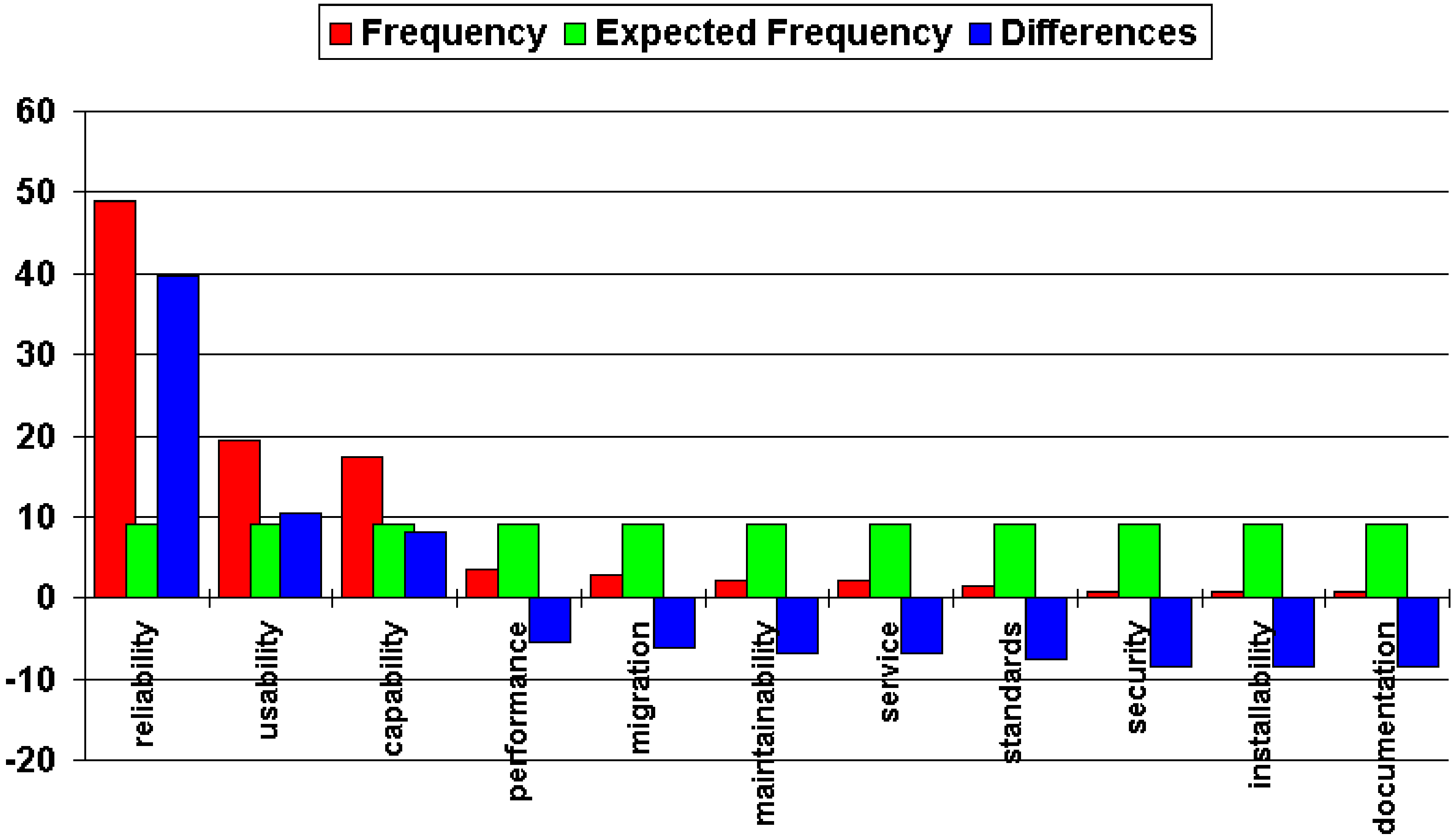
- Data Collection and Classification
  - *Core = <u>Orthogonal Defect Classification</u> scheme*
    - classification scheme based on defect attributes that are “orthogonal”
    - attribute values = spanning set
  - *“Classic” ODC attributes*
    - trigger, impact type, defect type, defect source

What is Traditional ODC?

- Trigger ("test case")
  - *workload/stress*
  - *normal mode*
  - *recovery/exception*
  - *startup/restart*
  - *hardware configuration*
  - *software configuration*

- Impact Type (on the user)
  - *usability*
  - *performance*
  - *reliability*
  - *installation*
  - *migration*
  - *documentation*
  - *serviceability*
  - *security*
  - *compatibility/co-existence*
  - *data/content*
  - *capability (function)*

Impact


Frequency
Expected Frequency
Differences
60
50
40
30
20
10
0
-10
-20
reliability
usability
capability
performance
migration
maintainability
service
standards
security
installability
documentation

What is Traditional ODC?

- Defect Type
  - *assignment (missing/incorrect)*
  - *parameter validation (m/i)*
  - *conditional logic (m/i)*
  - *timing/serialization (m/i)*
  - *data structure (m/i)*
  - *algorithm (m/i)*
  - *interface (m/i)*
  - *function (m/i)*
  - *build/merge (m/i)*
  - *(internal) documentation (m/i)*

- Defect Source
  - *stubbed code*
  - *re-fixed code*
  - *fixed code*
  - *re-written code*
  - *changed new code*
  - *new code*
  - *re-used code*
  - *third-party code*
  - *base code*

What is Traditional ODC?

- Other Useful Attributes
  - defect severity
  - number of source lines changed
  - when found (day, week, milestone #)
  - feature
  - source component or module
  - module complexity measures

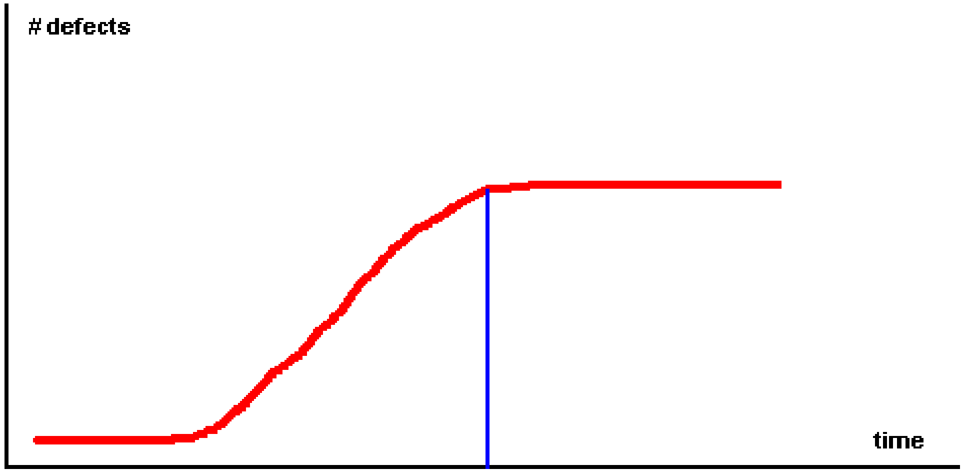
- Use Goal-Question-Metric (G-Q-M)

What is Traditional ODC?

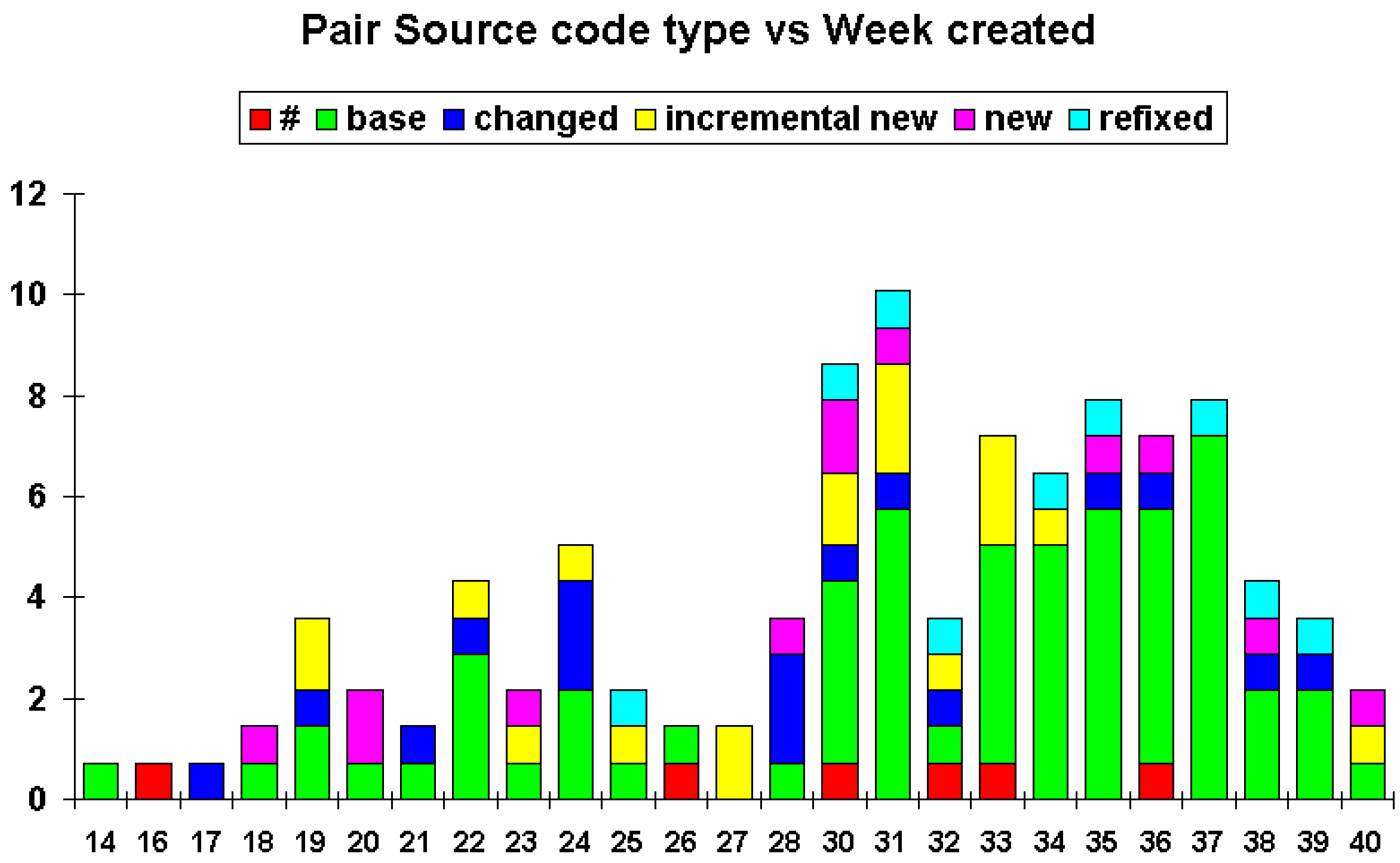
- Two Traditional Applications
  - *Verification method for defect growth curves*
  - *A feedback loop for software development projects who want to improve the way they develop software*

What is Traditional ODC?

- Verification method for defect growth curves



  - *stratify defects under the curve by "type"*
  - *verification that a project has actually reached a chronological milestone*

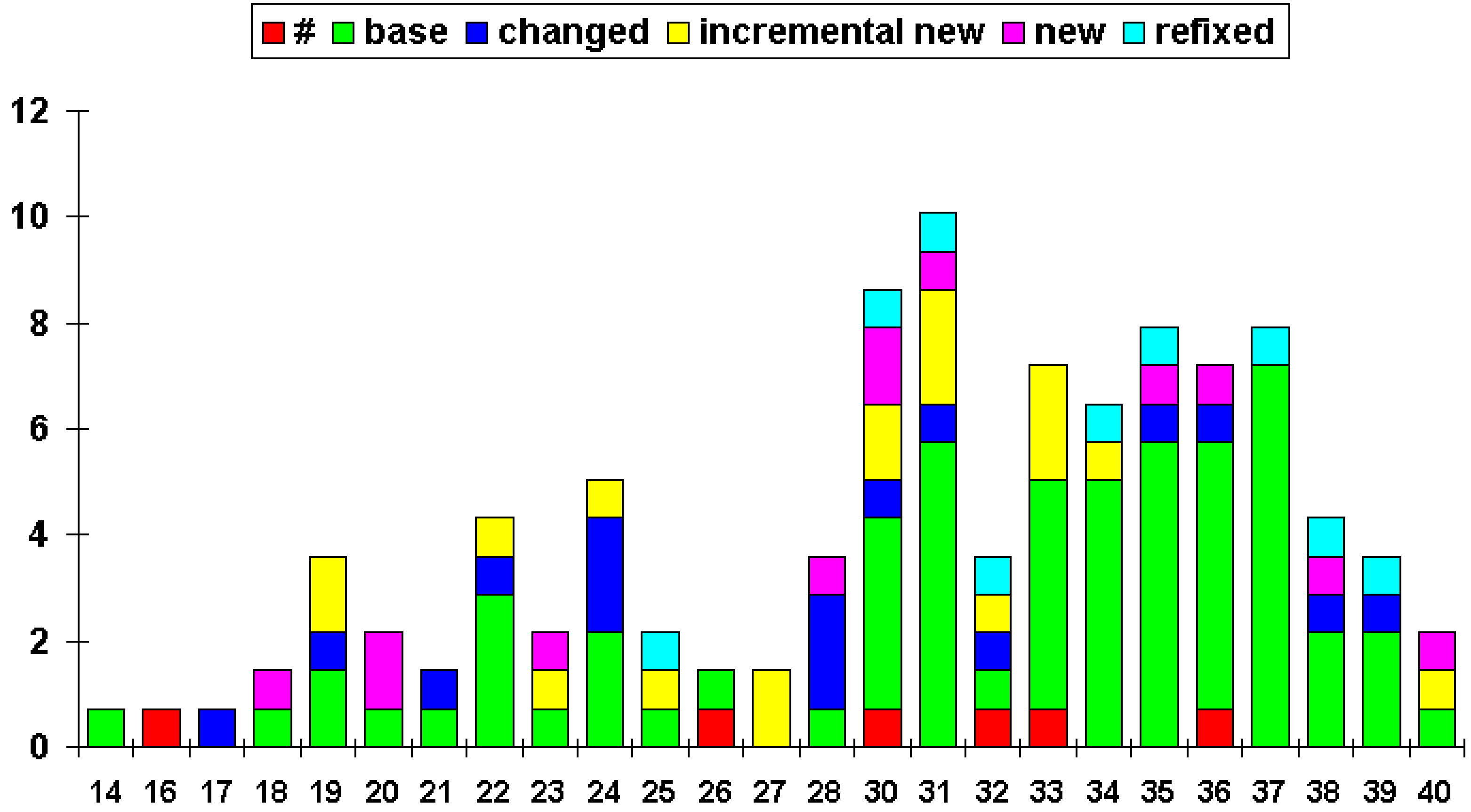

Pair Source code type vs Week created
#
base
changed
incremental new
new
refixed
0
2
4
6
8
10
12
14
16
17
18
19
20
21
22
23
24
25
26
27
28
30
31
32
33
34
35
36
37
38
39
40

What is Traditional ODC?

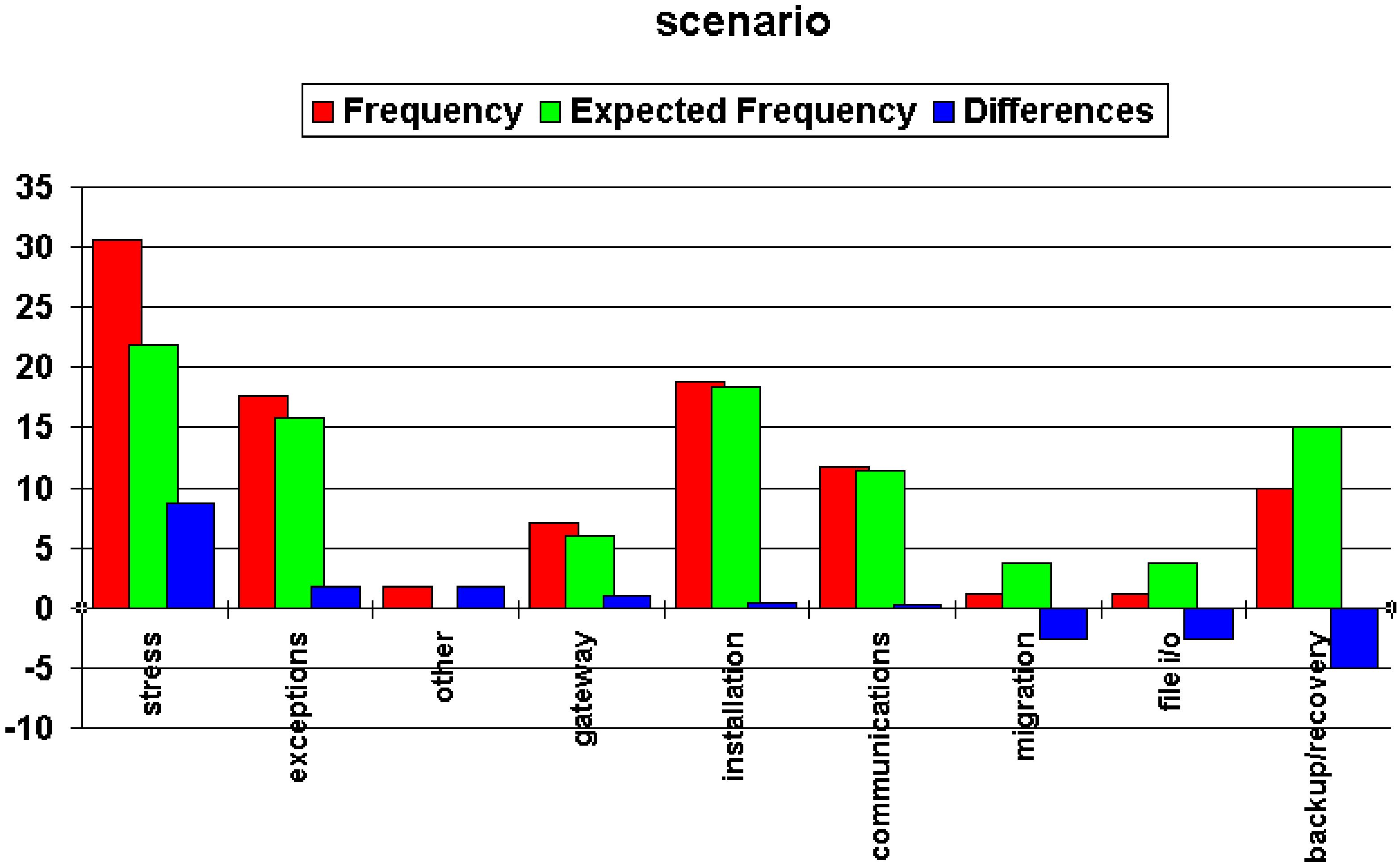
- Helping a team improve the way they develop software
- Milestone’s artifacts = code + defects
    - How can the number of new defects be reduced?
    - How can a team be more effective in detecting new or existing defects?
    - How can a team be more effective in fixing (removing) defects once they’re found?
    - How can defects be prevented in future cycles?

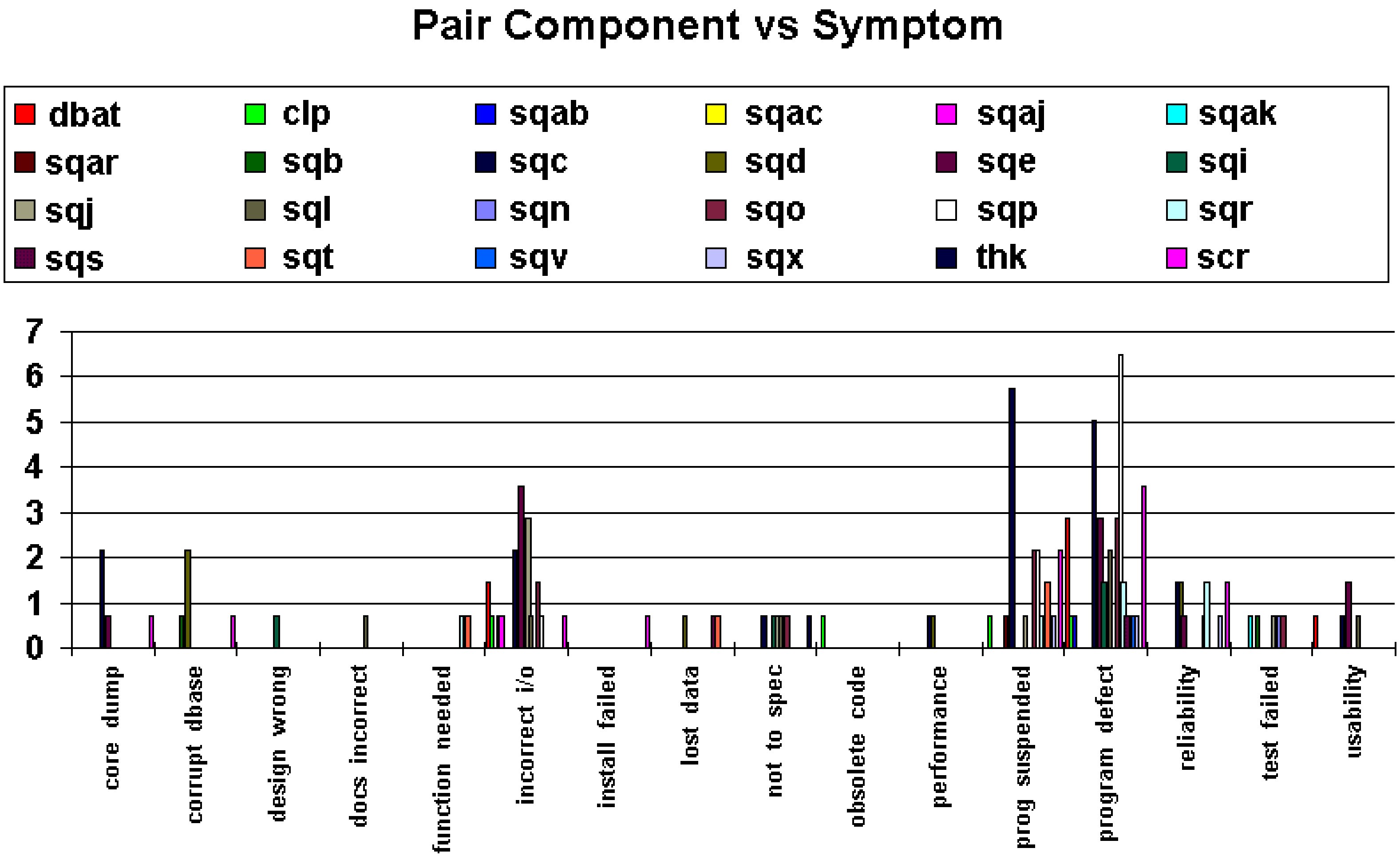
Attribute Focusing

- Data Mining: “Interestingness”
  - *Cases with greatest +/- differences between actual and expected frequencies*
  - *Apply to both single attribute-values and attribute-value pairs data*
- Ability to reduce the 100’s or 1000’s of attribute-value pairs to the 10-12 most interesting charts
- Represents the dominant trends in the data



Pair Source code type vs Week created


scenario
Frequency
Expected Frequency
Differences
35
30
25
20
15
10
5
0
-5
-10
stress
exceptions
other
gateway
installation
communications
migration
file i/o
backup/recovery


Pair Component vs Symptom
dbat
clp
sqab
sqac
sqaj
sqak
sqar
sqb
sqc
sqd
sqe
sqi
sqj
sql
sqn
sqo
sqp
sqr
sqs
sqt
sqv
sqx
thk
scr
0
1
2
3
4
5
6
7
core dump
corrupt dbase
design wrong
docs incorrect
function needed
incorrect i/o
install failed
lost data
not to spec
obsolete code
performance
prog suspended
program defect
reliability
test failed
usability

Recommendations

- More effective causal analysis => track more fields in RAID
- Make a few, good choices (“orthogonality”)
- Clear understanding of RAID fields and field values
  - *Field = A Question   Field values = Answers*
  - *Need a clear sense of the question being asked by a field and the meanings of the field values*
- Use G-Q-M

Extending ODC Concepts to Testing Readiness

- Knowing when to test a feature
- The right tests to run at the right time
- Understanding the completeness, stability and testability of a code base
  - *How can ODC concepts be applied to code artifacts to determine the completeness, stability and testability of a code base?*

Extending ODC Concepts to Testing Readiness

- Goal:

  Executing the right tests at the right time

- Question:

  When is this feature ready to test?

- Metrics:

  Source code change attributes

Extending ODC Concepts to Testing Readiness

- Object of study: (source) code
- Artifact: code change events
- Orthogonal Code Change Classification (OC^3)
  - *type - initialization, algorithm, parameters*
  - *source - new, changed, fixed, third-party*
  - *trigger - a feature or a fix*
  - *size and distribution*

Extending ODC Concepts to Testing Readiness

- Scenario: how would this work?
  - *Data Classification*
    - Developers write new feature code, change old base code and fix bugs (code change events)
    - Automatic classification using AST/Vulcan tools to extract the attribute values for each code change event
    - Bigger goal: Link the individual code change events back to a feature or bug - how?

Extending ODC Concepts to Testing Readiness

- Scenario (con’t)
  - *Data mining and Interpretation*
    - Explicit/direct, rule-based methods
    - Example: a feature is ready for testing when:
      - new code for a feature is detected
      - # lines of new code levels off
      - primary change trigger is not “feature”
      - change type doesn’t indicate any problems

Extending ODC Concepts to Testing Readiness

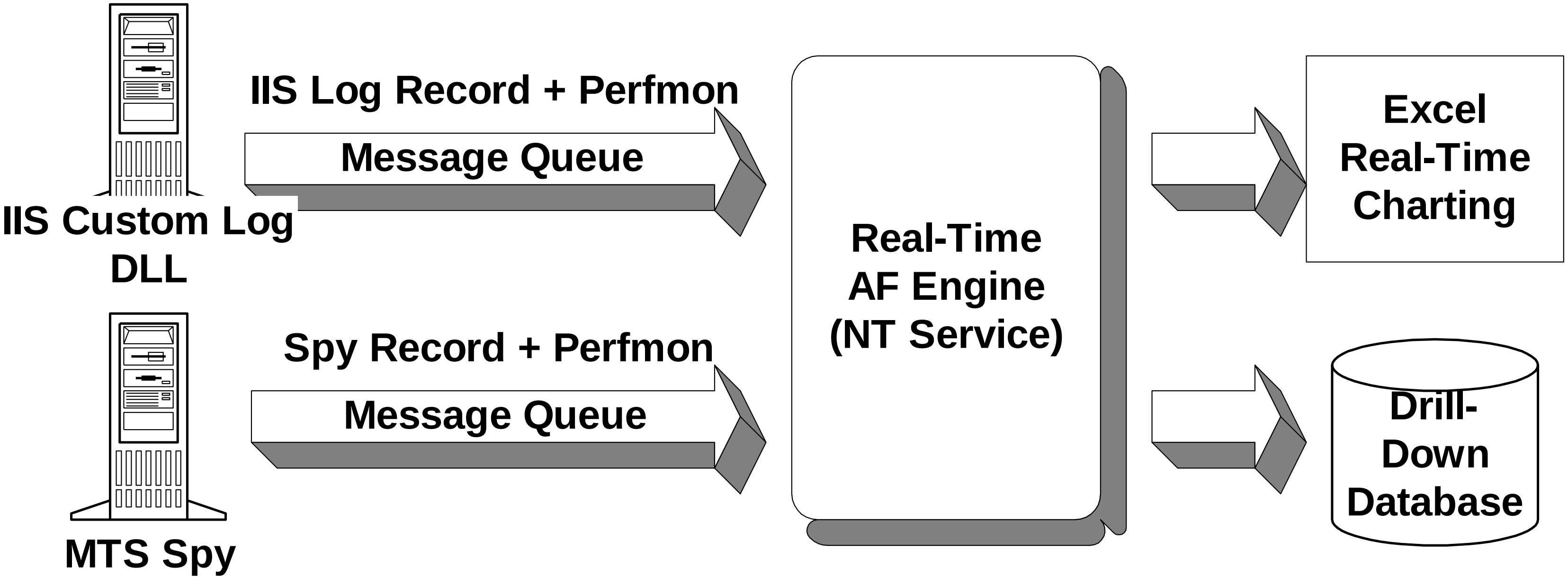
- Scenario (con’t)
  - *Ways to Improve the Rules Used:*
    - a) Attribute focussing
      - Project focus (all features: identify good/bad trends)
      - Feature focus (testability of a specific feature)
    - b) G-Q-M
    - c) Bayesian techniques?
  - *Develop action plans*
    - Which tests to run when?

Next Steps?

- Use Traditional ODC
  - Opportunity = a project in need of help (e.g. near code complete) -- use post classify a defect sample
  - New projects: customize RAID fields and values
- Tools for Test Decision Support
  - How can AST/Vulcan be used to derive attribute values for (source) code change events?
  - Develop rules database for determining test strategy
- Build Basic AF analysis and charting tool

Thank you.

AF Applied to Real-time Performance Diagnosis



IIS Custom Log DLL

IIS Log Record + Perfmon

Message Queue

MTS Spy

Spy Record + Perfmon

Message Queue

Real-Time AF Engine (NT Service)

Excel Real-Time Charting

Drill-Down Database