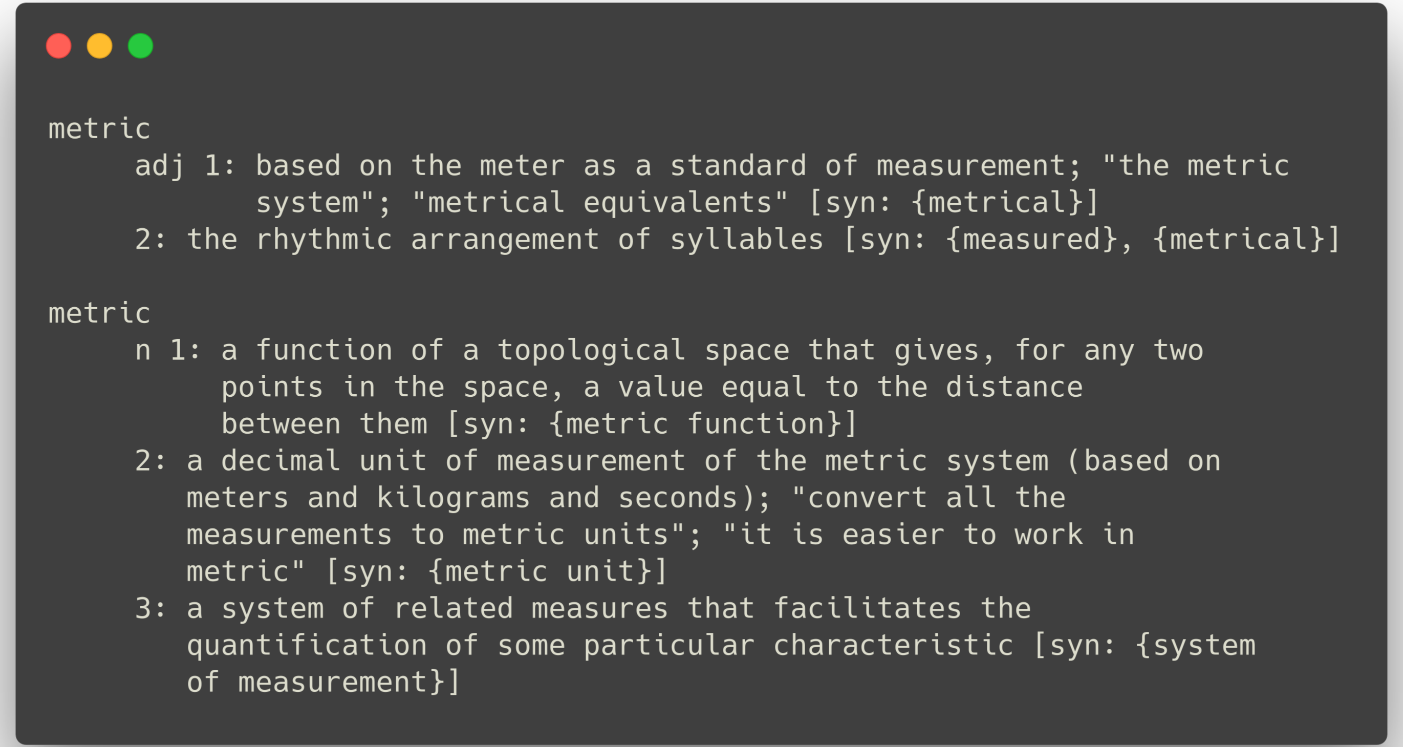
Actionable metrics are better metrics

+ Evaluating complexity, code churn, and developer activity metrics as indicators of software vulnerabilities

Presentation By Wenhan Zhu (Cosmos)

Metric



```
metric
     adj 1: based on the meter as a standard of measurement; "the metric
            system"; "metrical equivalents" [syn: {metrical}]
     2: the rhythmic arrangement of syllables [syn: {measured}, {metrical}]

metric
     n 1: a function of a topological space that gives, for any two
          points in the space, a value equal to the distance
          between them [syn: {metric function}]
     2: a decimal unit of measurement of the metric system (based on
        meters and kilograms and seconds); "convert all the
        measurements to metric units"; "it is easier to work in
        metric" [syn: {metric unit}]
     3: a system of related measures that facilitates the
        quantification of some particular characteristic [syn: {system
        of measurement}]
```

Source: WordNet

Metrics

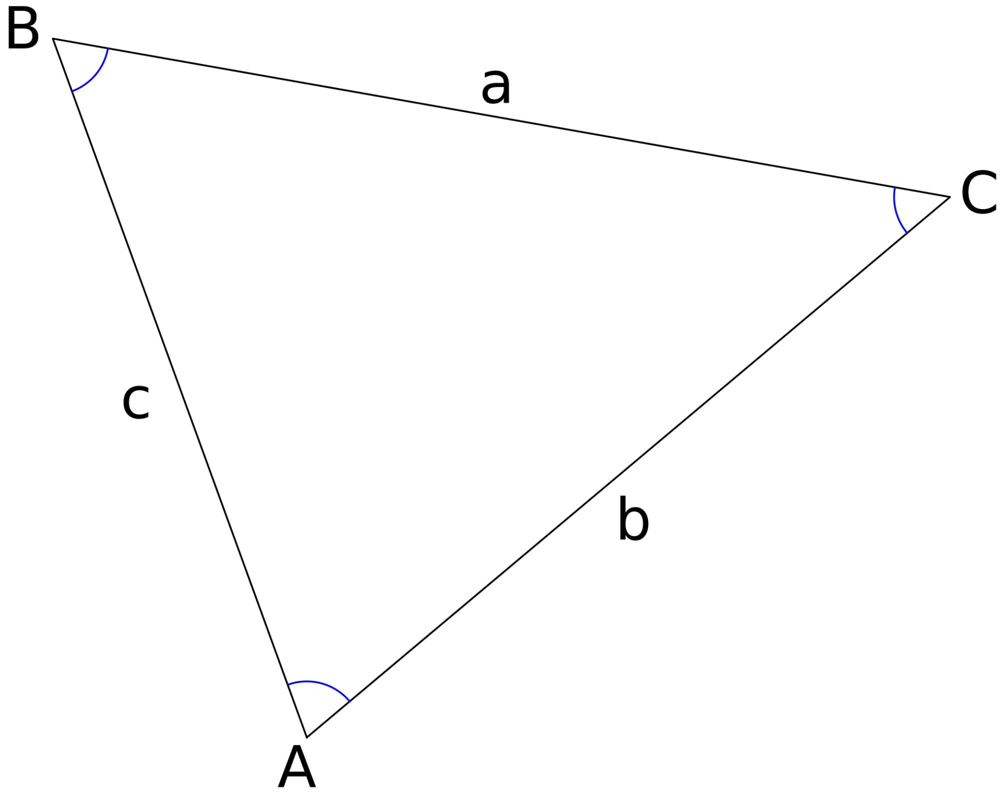
Does it matter?

Metrics

Does it matter?

Consider the following statement

In a triangle, the three interior angles always add to 180°



Metrics

Does it matter?

Consider the following statement

In a triangle, the three interior angles always add to 180°

Well... It's only true if we have the following metric tensor and distance function.

$$(ds)^2 = (dx)^2 + (dy)^2$$

$$d((x_1, y_1), (x_2, y_2)) = \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2}$$

Metrics

Does it matter?

Consider the following statement

In a triangle, the three interior angles always add to 180°

However, it's not **ALWAYS** true.

Poincaré half-plane

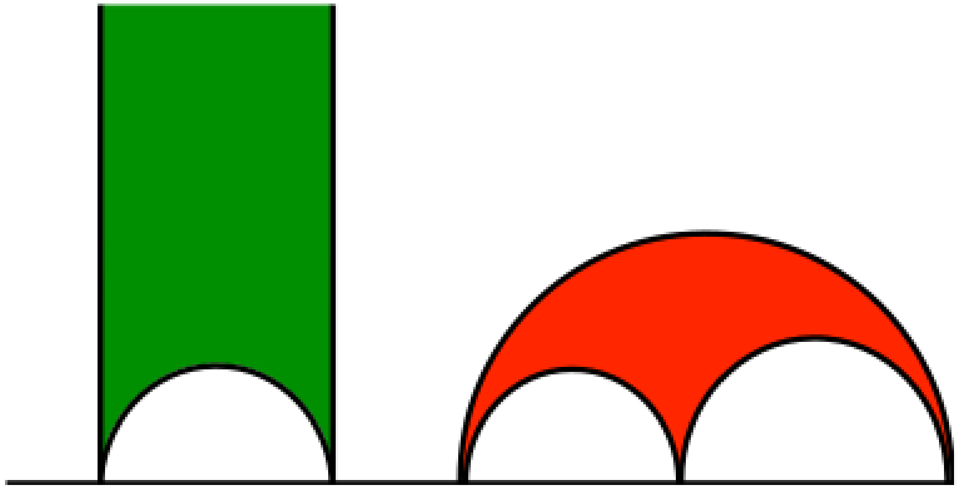
In the upper half plane of $\mathbb{R}$ defined as $H\{(x,y)|y>0;x,y\in\mathbb{R}\}$

With the following metric tensor.

$$(ds)^2 = \frac{(dx)^2+(dy)^2}{y^2}$$

which means a distance function of (x_1, y_1) and (x_2, y_2)

$$2\ln\frac{\sqrt{(x_2-x_1)^2+(y_2-y_1)^2}+\sqrt{(x_2-x_1)^2+(y_2+y_1)^2}}{2\sqrt{y_1 y_2}}$$



Source: Wikipedia

Metrics

Two types of metrics in SE

- Actionable Metrics
- Emergent Metrics

Emergent Metrics

Metrics that provides overall assessment of the data but can't be relied upon to diagnose the problem.

Examples:

- Number of Bugs
- Code Churn

Actionable Metrics

Metrics that measures a property directly under a developer's control.

Examples:

- Number of developers working on a file
- Number of method callees
- Number of parameters

CCD

CCD Metrics as Indicators of Software Vulnerabilities

CCD:

- Complexity
- Code Churn
- Developer Activity

CCD

Complexity

- IntraComplexity
  - Line of Code (LOC)
  - Number of functions defined in a file
  - # LOC devoted to declarations
  - # LOC devoted to preprocessing
  - Sum of essential complexity
  - Sum/Max of strict cyclomatic complexity
  - Sum/Max of max nesting level of control constructs
- Coupling
  - Sum/Max of inputs to a function
  - Sum/Max of assignments to the parameters to call
- Comments
  - Ratio of comments to code

Code Churn

In general means the measure of the rate code evolves.

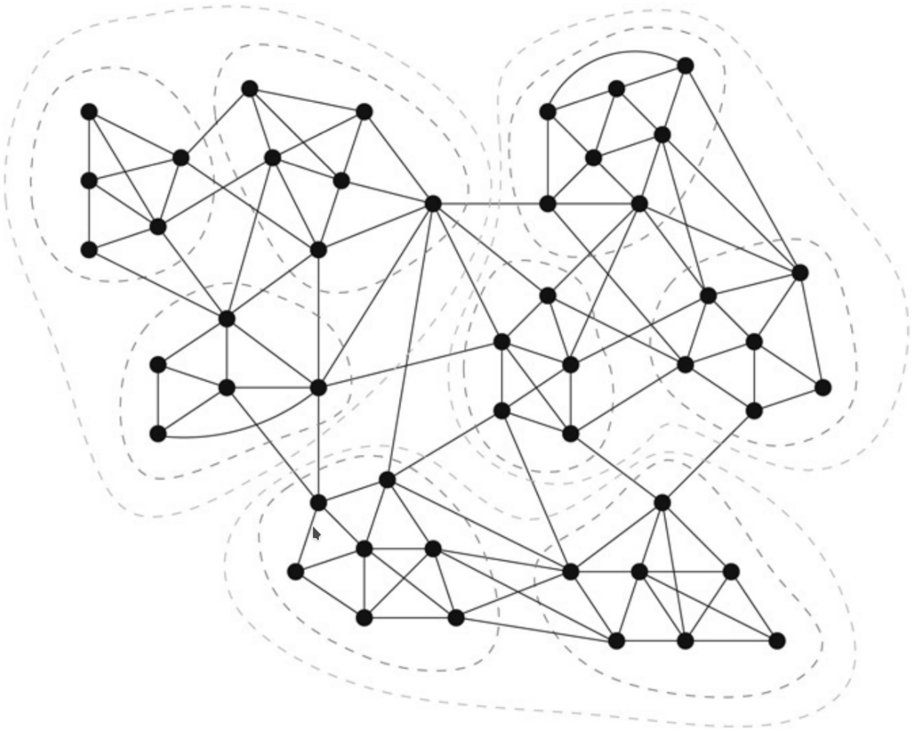
- # of check-ins for a file since creation
- Cumulated number of code lines changed since creation
- Cumulated number of new code lines since creation

Developer Activity

Developer Network

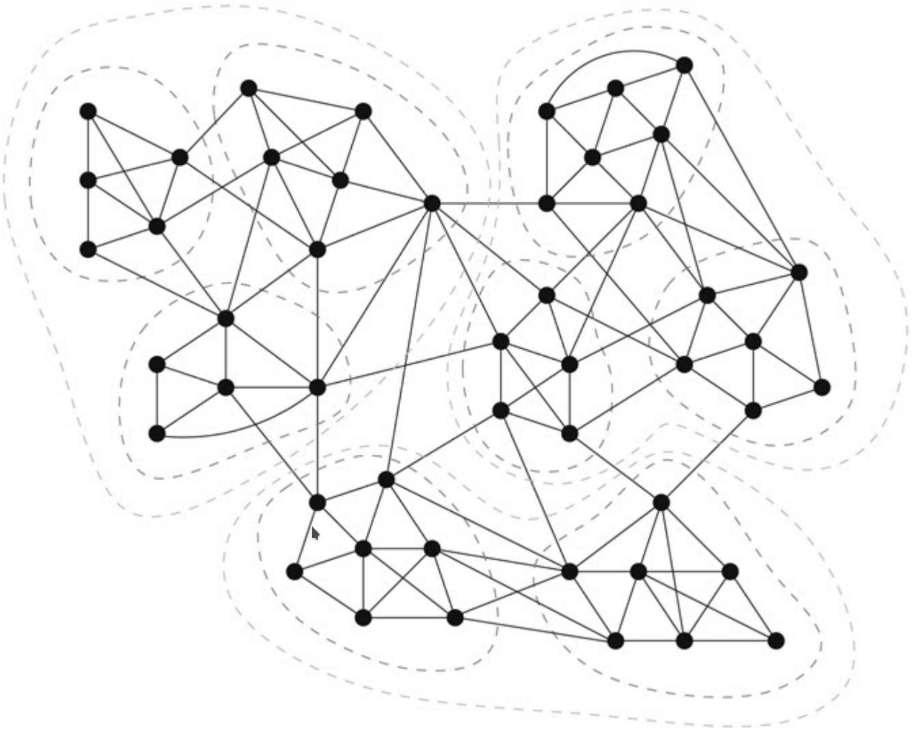
Consider each developer as a node in a graph.

When 2 developers worked on the same file, there's an edge between the two developers.



Centrality

- degree: number of neighbours of a node
- closeness: average distance from a node v to any other node that it can reach
- between: number of geodesic paths that include v



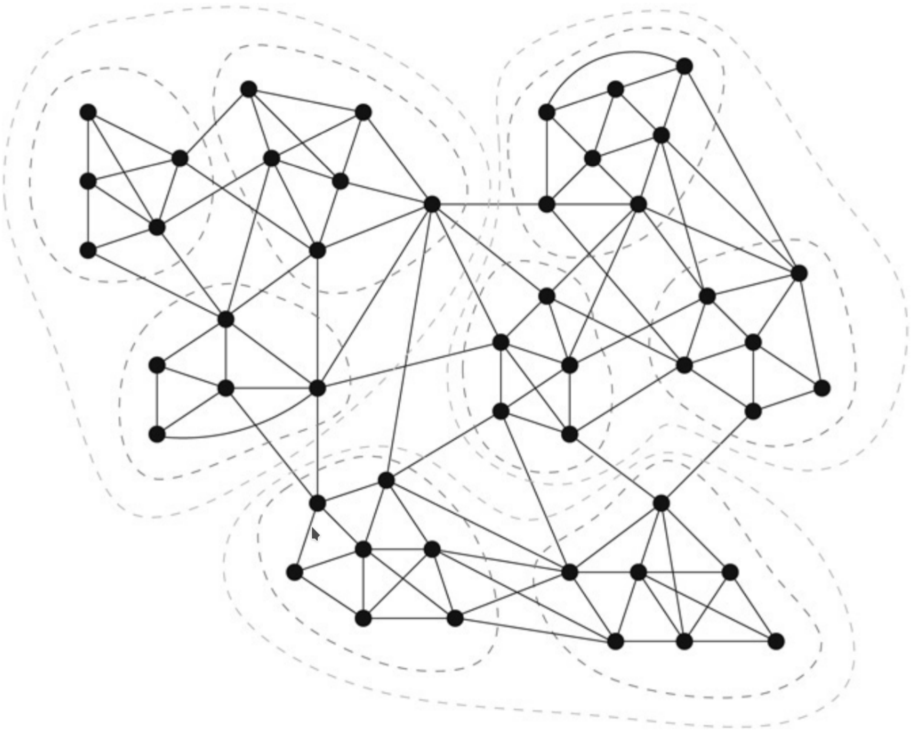
Developer Activity

Cluster

A set of nodes such that there are more edges within a set than edges between a set and other set of nodes

Edge Betweeness

Number of geodesic path though a edge



Developer Activity

Contribution Network

Bipartite graph with files and developers as nodes.

If a developer edited a file, there is an edge between them.

Developer Activity

- Centrality
    - Min/Avg degree
    - Max/Avg closeness
    - Min/Avg betweeness
- Cluster
    - Max/Avg Edge Betweeness
- Contribution Centrality
    - # Devs
    - Contribution Network Closeness

Metrics to measure

TP, TN, FP, FN

- Probability of Detection (PD)

$$PD = TP/(TP + FN)$$

- Probability of False alarm (PF)

$$PF = FP/(FP + TN)$$

- Precision (P)

$$P = TP/(TP + FP)$$

Metrics to measure continued

- File Inspection (FI)

$$FI = (TP + FP)/(TP + TN + FP + FN)$$

- LOC Inspection (LI)

$$LI = (TP_{LOC} + FP_{LOC})/(TP_{LOC} + TN_{LOC} + FP_{LOC} + FN_{LOC})$$

- File Inspection Reduction (FIR)

$$FIR = (PD - FI)/PD$$

- LOC Inspection Reduction (LIR)

$$LIR = (PV - LI)/PV$$

- Predicted Vulnerability (PV)

$$PV = TP_{Vuln}/(TP_{Vuln} + FN_{Vuln})$$

Experiments

Data is skewed and only a extremely small percentage (<1.5%) have vulnerabilities.

Undersampling was used.

Case studies

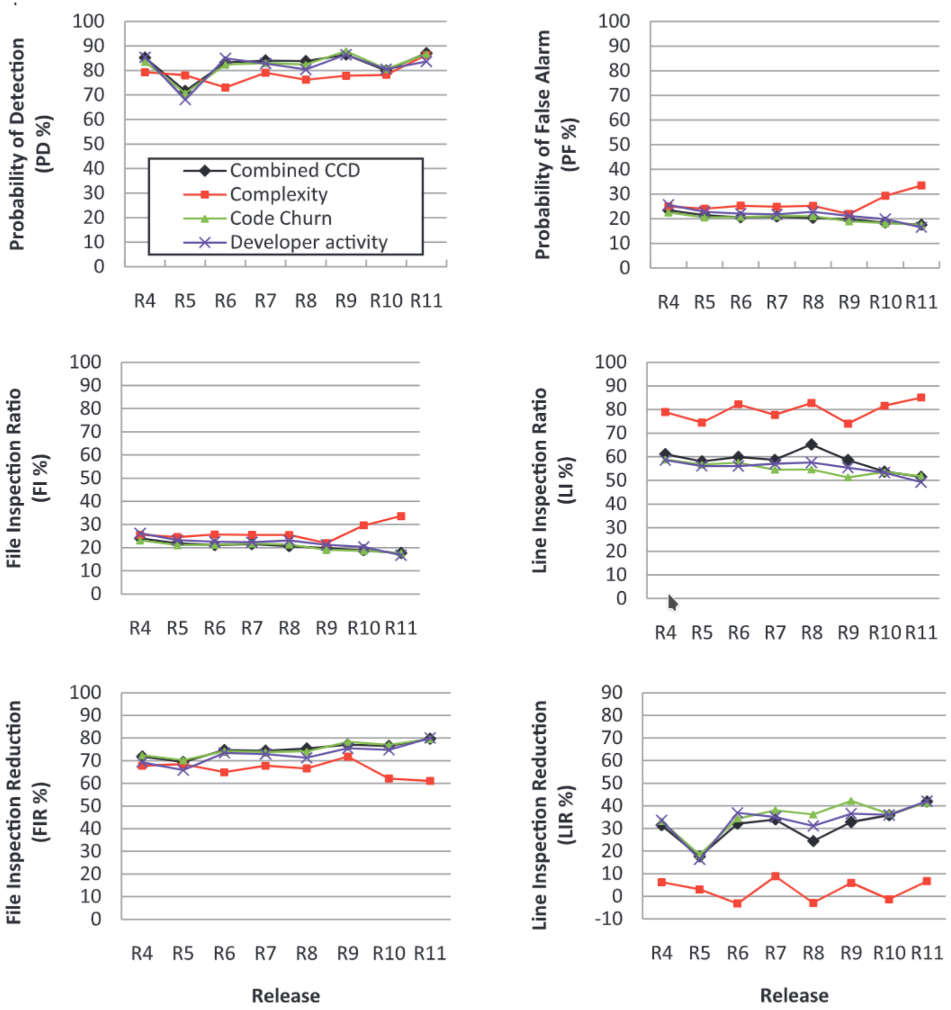
- Firefox

  Merge 3 version for the following version

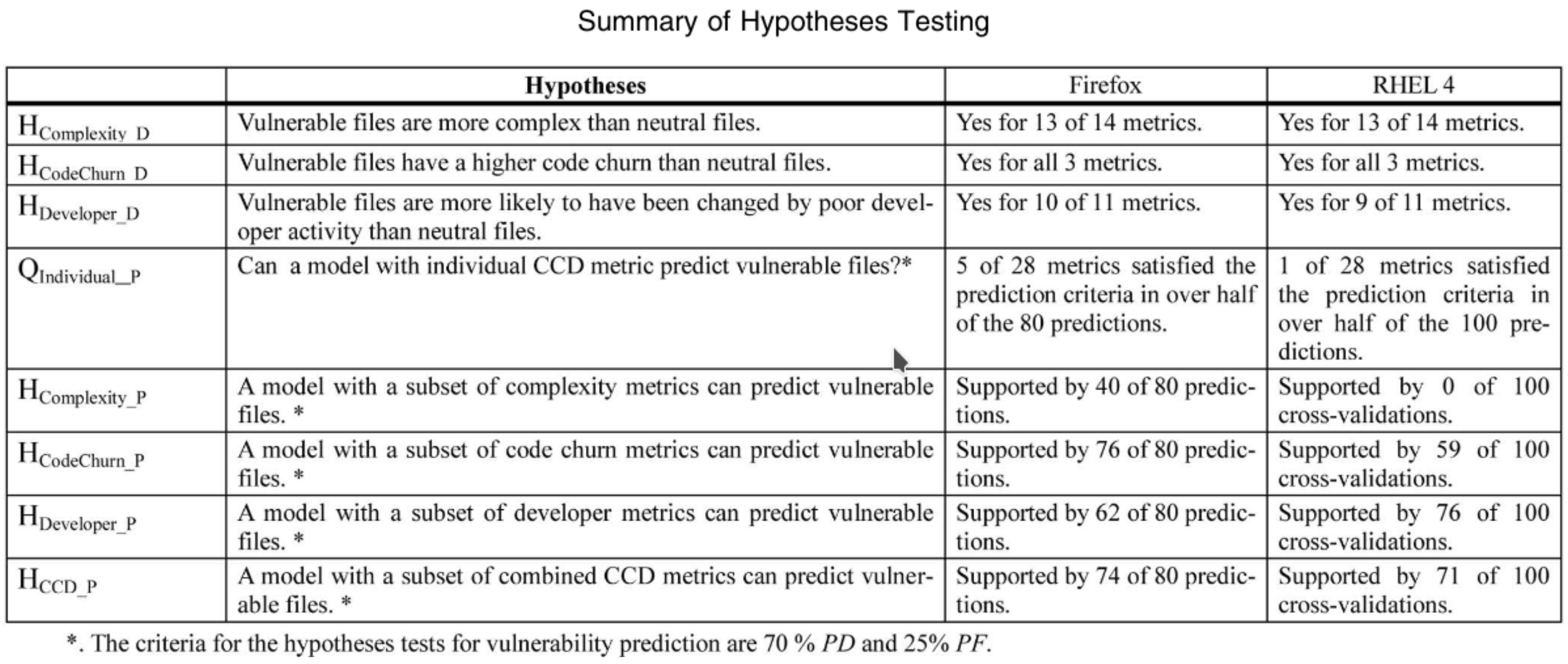
- RHEL 4

  Used bug reports from Red Hat and actual code from Linux kernel

Firefox


Probability of Detection (PD %)
Combined CCD
Complexity
Code Churn
Developer activity
Probability of False Alarm (PF %)
File Inspection Ratio (FI %)
Line Inspection Ratio (LI %)
File Inspection Reduction (FIR %)
Line Inspection Reduction (LIR %)
R4 R5 R6 R7 R8 R9 R10 R11
0 10 20 30 40 50 60 70 80 90 100
-10
Release

Results & Conclusions



Summary of Hypotheses Testing

	Hypotheses	Firefox	RHEL 4
$H_{\text{Complexity\_D}}$	Vulnerable files are more complex than neutral files.	Yes for 13 of 14 metrics.	Yes for 13 of 14 metrics.
$H_{\text{CodeChurn\_D}}$	Vulnerable files have a higher code churn than neutral files.	Yes for all 3 metrics.	Yes for all 3 metrics.
$H_{\text{Developer\_D}}$	Vulnerable files are more likely to have been changed by poor developer activity than neutral files.	Yes for 10 of 11 metrics.	Yes for 9 of 11 metrics.
$Q_{\text{Individual\_P}}$	Can a model with individual CCD metric predict vulnerable files?*	5 of 28 metrics satisfied the prediction criteria in over half of the 80 predictions.	1 of 28 metrics satisfied the prediction criteria in over half of the 100 predictions.
$H_{\text{Complexity\_P}}$	A model with a subset of complexity metrics can predict vulnerable files. *	Supported by 40 of 80 predictions.	Supported by 0 of 100 cross-validations.
$H_{\text{CodeChurn\_P}}$	A model with a subset of code churn metrics can predict vulnerable files. *	Supported by 76 of 80 predictions.	Supported by 59 of 100 cross-validations.
$H_{\text{Developer\_P}}$	A model with a subset of developer metrics can predict vulnerable files. *	Supported by 62 of 80 predictions.	Supported by 76 of 100 cross-validations.
$H_{\text{CCD\_P}}$	A model with a subset of combined CCD metrics can predict vulnerable files. *	Supported by 74 of 80 predictions.	Supported by 71 of 100 cross-validations.

*. The criteria for the hypotheses tests for vulnerability prediction are 70 % *PD* and 25% *PF*.

Summary

Hightlights

- Most metrics work 24/28 for both
- *CountLinePreprocessor* and *CommentDensity* were not discriminative between neutral and vulnerable files
- *DNMinDegree* negatively correlated with vulnerability while *DNAvgDegree* positively correlated in both.
- *DNAvgBetweenness* disagrees with hypothesis and unable to find clear reason.
- Precision was low

Discussion

- Vulnerability vs Fault?
- What are some questionable metrics you've seen in real life?
- Almost a decade after the publication of this paper, have we improved on the metrics we use in SE?

Thank you!

Slides powered by Remark