Finding Dependencies from Defect History

Rajiv Das, Wipro Technologies
Jacek Czerwonka, Microsoft Corporation
Nachiappan Nagappan, Microsoft Corporation

Context – Windows Development

- Size and scope
  - 40+ MLOC
  - Development team spread all over the world
  - 1B+ users
  - 400,000 supported devices
  - 6,000,000 apps running on Windows
  - Up to 10 years of servicing
- Challenges
  - Large, complex codebase with millions of tests
  - Diverse customer base
  - Time and resource constraints
  - Diverse test execution
  - Very low tolerance to failure

Problems

- Unknown Dependencies
  - Static, Dynamic Analysis does not find everything
- Large number of Dependencies
  - How to prioritize Integration Testing when changes are routine and costs involved are high?

Motivation

Graphics Driver crashes whenever user 'Pastes' image to *Photo Editor*

- Internal defect in driver, exposed by unknown dependency between editor and driver

Defect Id	2125
Title	Graphics driver crashes on Paste
Status	Closed
Opened By	Alice
Opened On	7-November-2005
Affected Component	Drivers\Video\Driver.sys
Resolution	Fixed
Resolved By	Bob
Resolved On	1-December-2005

Sample Defect Record

* Source Component Photo Editor, not recorded explicitly

Definitions

- **Dependency:** *If defects are found frequently in component C_1 when component C_2 is tested, then C_2 may be dependent on C_1*
- **Source Component:** *The component containing the defect*
- **Affected Component:** *The component affected due to a defect*

Frequent Itemset Mining

If X and Y are items in the transaction dataset:

- ***Support*(*X*)**: probability of occurrence of *X*, *p(X)*.
- ***Confidence*(*X*=>*Y*)** : how frequently *Y* occurs when *X* occurs, *p (Y|X).*
- ***Importance* (*X*=>*Y*)**: The log likelihood of *Y* occurring with *X*, than without it i.e. $log \frac{p(Y \mid X)}{p(Y \mid \text{not } X)}$

In a transaction dataset, frequent itemsets X and Y can be found using **Dependence Rules Mining**:

1. *Support(X)*, *Support (Y)* and *Support(X and Y)* above threshold
2. *Confidence(X=>Y)* above threshold

How to Identify Dependencies

- Let C_S be source component
- Let C_A be affected component
- Find frequent pairs of source and affected components in the component map using Dependence Rules, C_S => C_A where
  1. *Support*(C_A), *Support*(C_S), *Support*(C_A *and* C_S) >= support cutoff
  2. *Confidence*(C_S => C_A) >= confidence cutoff
  3. *Importance*(C_S => C_A) is positive, meaning that the affected component is positively statistically dependent on the source component

How to Rank Dependencies

- Rank dependencies first by confidence and then by importance.
  - Higher confidence has higher rank
  - Higher importance has higher rank
- For *k* topmost dependencies,
  - First chose all dependencies greater than confidence cutoff
  - Then choose *k* dependencies out of them with largest importance

Example

Affected	Source
C1	C2
C3	C2
C4	**C2**
C1	C3
C2	C5
C1	C2
C4	**C2**
C1	C2

Component Map

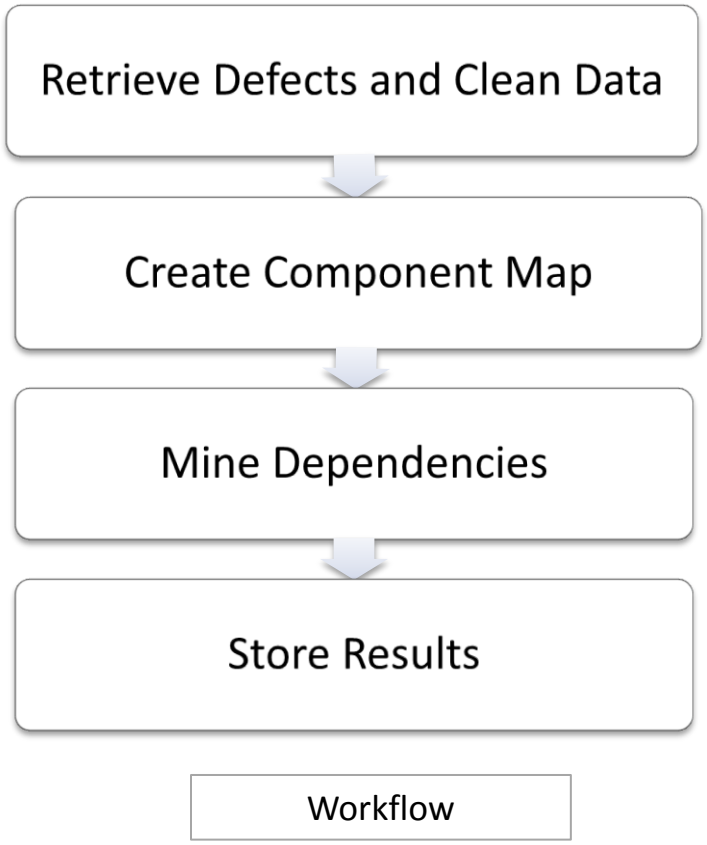
- **Support** 0.25
- **Confidence** 0.1
- **Importance** > 0

Rule	Support	Confidence	Importance
C2 => C4	2	2/6 = 0.33	0.176

Dependencies Found

Ladybug Tool

- Automates dependency discovery
- Easily Customizable
- Built on SQL Server Platform – Analysis Services, Integration Services, SQL Server



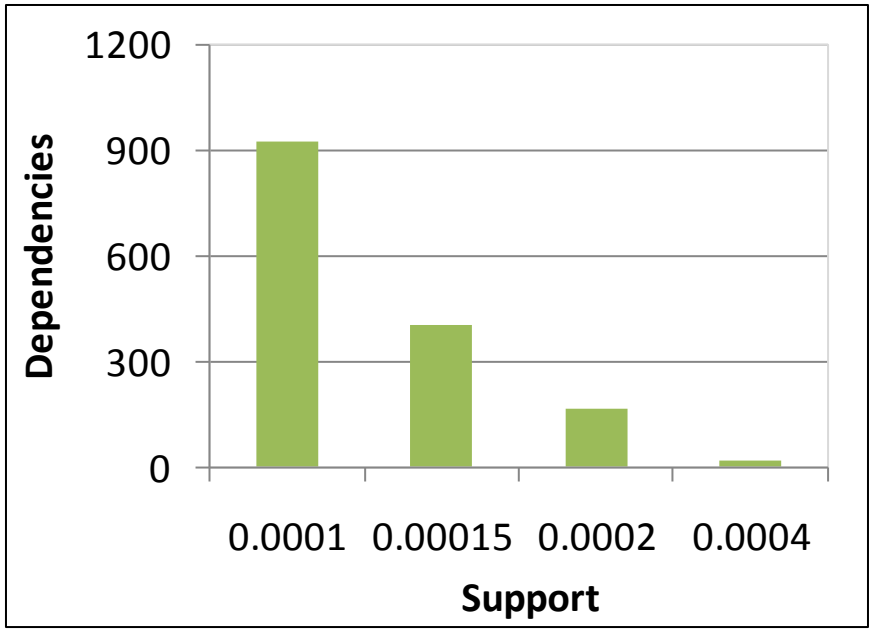
Experiment

- Pre-release Defects for Windows Vista and Windows Server 2008

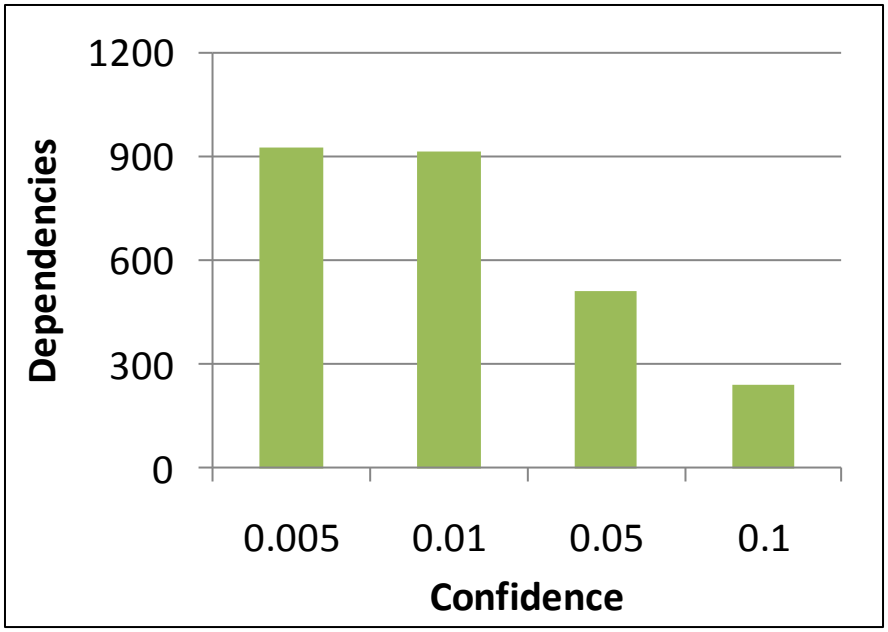
Size	92,976
Defects Included	28,762
Affected Components	1,649
Source Components	1,480

Input Component Map

Dependencies Found



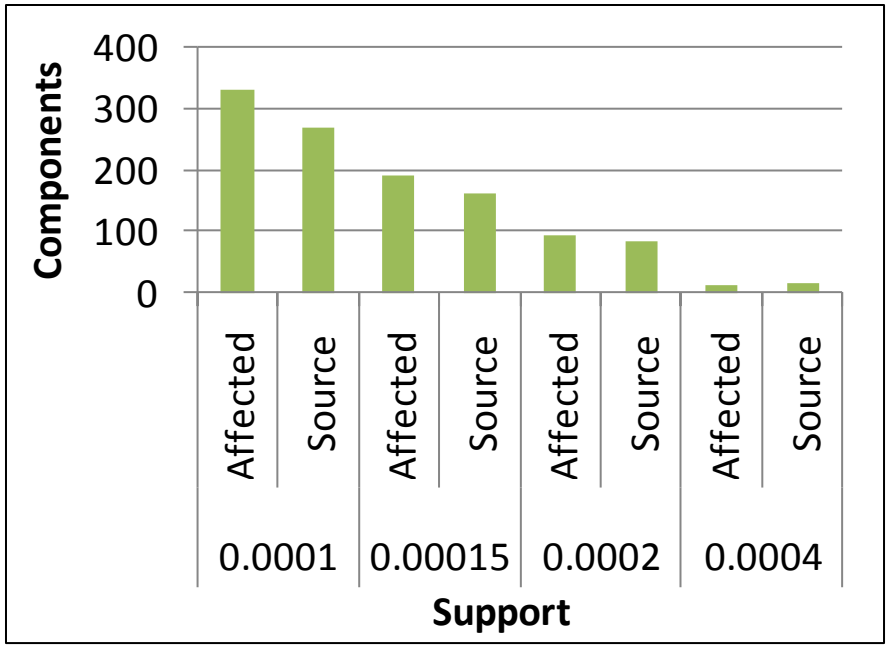
Dependencies found by varying Support with Confidence 0.005



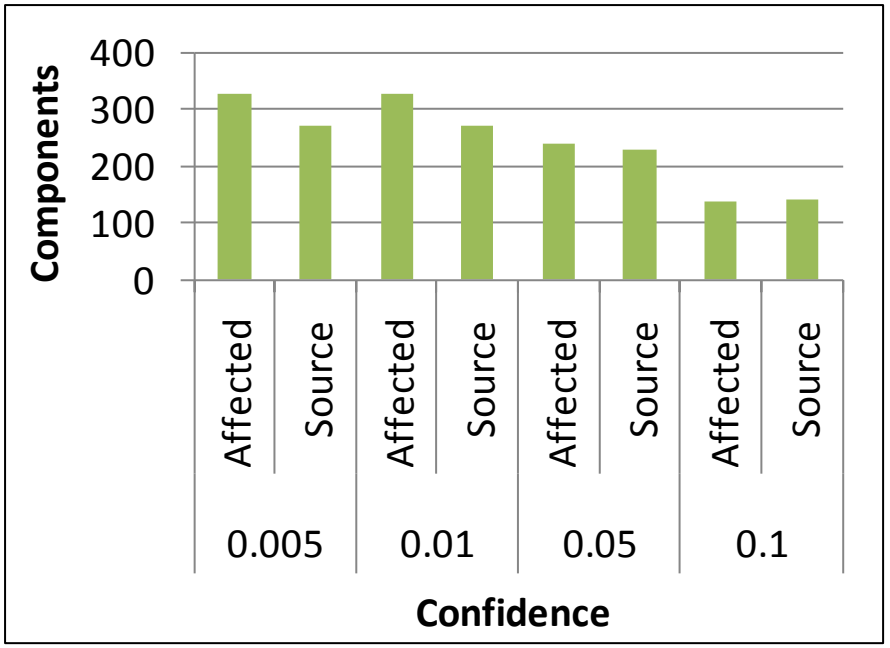
Dependencies found by varying Confidence with Support 0.0001

Outcome indicates that dependencies can be found for various combinations of support and confidence thresholds

Source and Affected Components



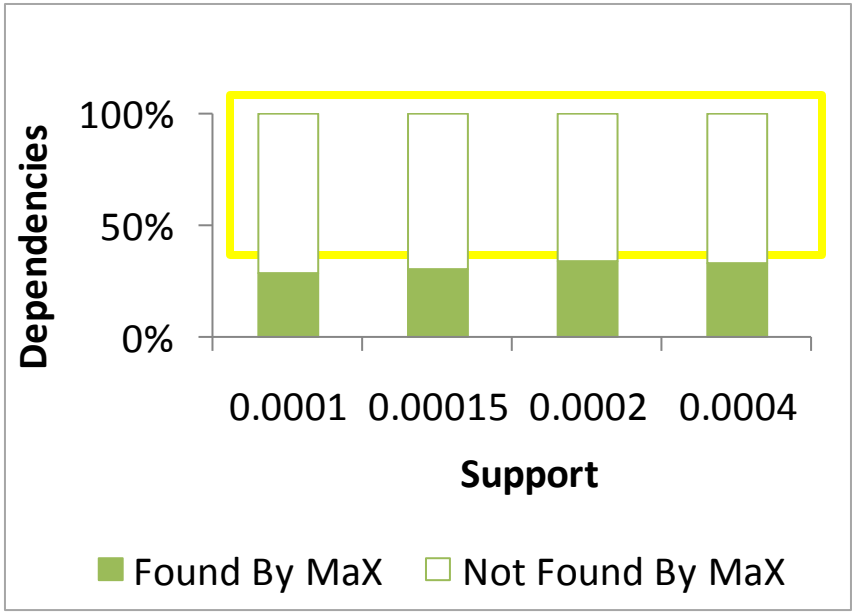
Source and Affected Components included in dependencies found by varying Support with Confidence 0.005



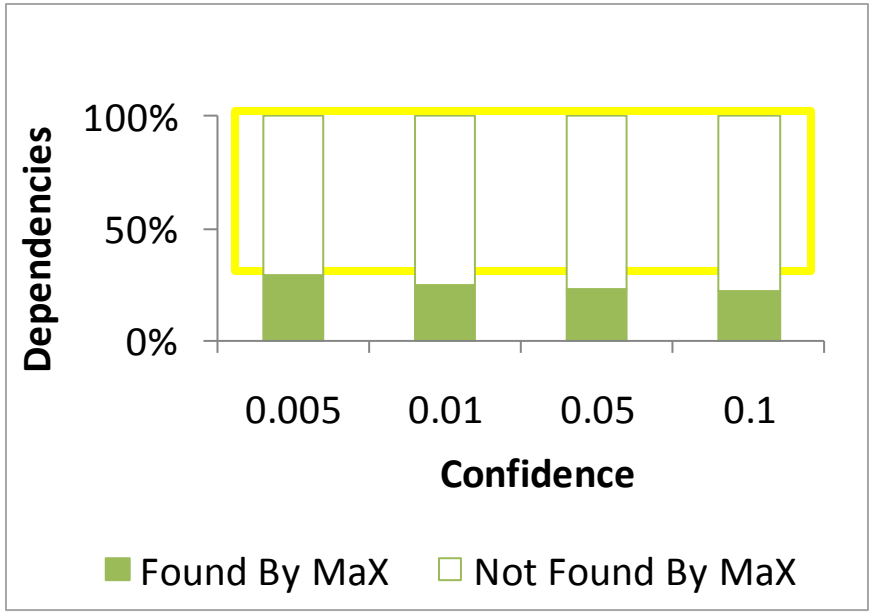
Source and Affected Components included in dependencies found by varying Confidence with Support 0.0001

Fewer components get included in the results as the thresholds are raised

Effectiveness



Comparison with MaX for dependencies found with different Support values with Confidence 0.005



Comparison with MaX for dependencies found different Confidence values with Support 0.0001

Outcome indicates our approach can possibly find new dependencies.

Manual Validation

Total dependencies	276
Votes Cast	211
Dependencies with Votes	182
Dependencies with Multiple Votes	29
Experts Invited	127
Experts Participated	**70**

Dependencies found by our method with support 0.0001 and confidence 0.005

Vote tally for different buckets of dependencies

In general, owners seemed to confirm the dependencies considerably more often than reject them

Manual Validation (2)

	Found by MaX	Not Found by Max	Row Total
Confirmed	27	**115**	142
Rejected	12	28	40
Total	39	143	182

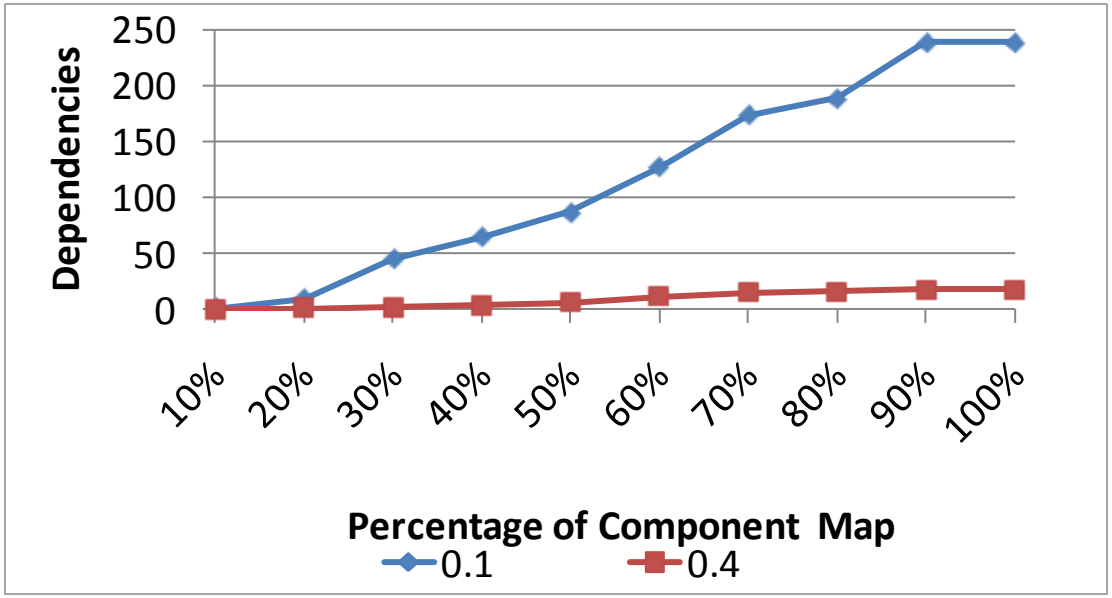
Contingency table showing votes versus detection by MaX

H_0: Experts voted based on the rule content and their background system knowledge independently of what the MaX data says, which they may have seen before

We do not reject H_0 at 95% confidence level using Chi-Square analysis.

It is possible to discover additional new important dependencies using our method

Applicability



Dependencies found as a function of defect reports, for different confidence values and support count 25, indicates more defect reports yielded more dependencies

We can start mining at early phase of software development and keep refining model over time as more data becomes available.

Alternative Input Component Maps

	Component Map 1	Component Map 2
Ownership Error (%)	0	5
Map Size	89,075	92,976
Defects	28,028	28,762
Affected Components	1,637	1,649
Source Components	1,470	1,480

- Alternative Component Maps were extracted using different values of Ownership Errors
- No noticeable difference in the results

* Detailed description available in paper

Threats to Validity

- Dependencies cannot be found for Components that have not been part of significant number of defects in the past
- Our study is on a well-componentized, large-scale software system with a stable development process and considerable number of defect reports.
- For practical applications, it may be useful to use higher thresholds to restrict the outcome to the most significant rules only.

Conclusions

- New Approach to *identified* software dependencies
- An approach to *rank dependencies* using defect history
- *Ladybug tool* to mine defect history for new dependencies
- Possible to *start mining* at any phase of development and refine models over time
- *Found a large number of dependencies* confirmed by experts but are not found by static analysis tools
- Ladybug analysis has been incorporated in a larger change analysis and test targeting system used in Windows Serviceability and recommendations are used by hundreds of engineers every month

Future Work

- Apply Ladybug to defect datasets of other software
- Look at ways of incorporating user judgment to generate better dependency recommendations

Thank You